RL-TR-97-174
Final Technical Report
October 1997

AN INDEPENDENT EVALUATION OF THE ROME LABORATORY FRAMEWORK FOR CERTIFICATION OF REUSABLE SOFTWARE COMPONENTS

Underwriters Laboratories, Inc.

Charlotte O. Scheper, Janet S. Flynt, Sharon E. Smith,
Cleo J. Jones, and Jeffrey A. Torres

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Rome, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-97-174 has been reviewed and is approved for publication.

APPROVED: Deborah A. Cerino

DEBORAH A. CERINO
Project Engineer

FOR THE DIRECTOR:

WARREN H. DEBANY, JR., Technical Advisor
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL/C3CB, 525 Brooks Rd, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Oct 97	Final Jun 95 - May 97

4. TITLE AND SUBTITLE
AN INDEPENDENT EVALUATION OF THE ROME LABORATORY FRAMEWORK FOR CERTIFICATION OF REUSABLE SOFTWARE COMPONENTS

5. FUNDING NUMBERS
C - F30602-95-C-0128
PE - 63728F
PR - 2527
TA - 02
WU - 37

6. AUTHOR(S)
Charlotte O. Scheper, Janet S. Flynt, Sharon E. Smith, Cleo J. Jones and Jeffrey A. Torres

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Underwriters Laboratories, Inc.
12 Laboratory Drive
PO Box 13995
Research Triangle Park, NC 27709-3995

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory/C3CB
525 Brooks Rd
Rome, NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-97-174

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Deborah A. Cerino, C3CB, 315-330-2054

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE
N/A

13. ABSTRACT *(Maximum 200 words)*

In this effort, Underwriters Laboratories (UL) conducted an independent review of the Rome Laboratory framework for certification of reusable software components from the perspective of its potential application by a third party test laboratory. The review consisted of two parts: a desk review and a trial application. The desk review considered the goals and objectives of the framework, its costs and benefits, its standardization potential, and technology transfer issues. In the trial application, an asset was selected and the default certification process was applied. A scenario for the development and use of the asset was created to provide the context from which certification concerns, criteria, and requirements were identified. Benchmark components, including instrumented code, models and fault sets, were created for the asset to provide expected results to access the results of the procedures specified by the default process. UL determined the framework is built upon clear and sound research. UL found that the framework's effort to associate techniques with defect types is a necessary approach, however, they recommended more detailed analysis of the relationships between techniques and defects before techniques can be recommended with confidence. UL plans to pursue this research in an experimental lab being established at UL. They plan to, over time, broaden the framework's certification model and position components of the framework for incorporation into certification standards.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Software Certification, Software Assessment, and Evaluation	72
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

List of Tables

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States Government nor Underwriters Laboratories Inc. nor any of their employees nor their contractors, subcontractors, or their employees makes any warrantee expressed or implied, or assumes any legal liability or responsibility for damages arising out of or in connection with the interpretation, application, or use of or ability to use any information, apparatus, product, or process disclosed, or represents that its use would not infringe on privately owned rights. This report may not be used in any way to infer or to indicate UL's endorsement of any product, or to infer or to indicate acceptability for Listing, Classification, Recognition, or Certificate Service by Underwriters Laboratories Inc. of any product or system.

Executive Summary

The Air Force Rome Laboratory is developing a framework for certification of reusable software components. In this effort, Underwriters Laboratories (UL) conducted an independent review of the framework from the standpoint of its potential future application by third party test laboratories. The review of the framework consisted of two parts: a desk review and a trial application. The desk review considered the goals and objectives of the framework, its costs and benefits, its standardization potential, and technology transfer issues.

In the trial application, an asset was selected and the default certification process was applied to it. A scenario for the development and use of the asset was created to provide the context from which certification concerns, criteria, and requirements were identified. Benchmark components, including instrumented code, models, and fault sets, were created for the asset to provide expected results to assess the results of the procedures specified by the default process.

Although care was taken to construct a framework that is usable and practical and the resulting Rome Laboratory Certification Framework is built upon clear, decisive, and sound research, there are market and technical issues that would prevent the direct implementation of the framework in a third party certification environment. It appears that the application of the evaluation procedures by a third party certifier to a component designed and produced for the commercial market may not be an appropriate application for the framework. It may be more practical for certifiers to use the framework to derive requirements for testing and evaluation of components that the developers would implement. The certifiers would then verify that the developers had implemented the specified activities.

In terms of cost and benefit, the benefits of certification could be demonstrated to outweigh its costs under the current business model, which is targeted towards traditional reuse repositories. However, a greater cost/benefit could be more immediately evident under other business models, such as the Third Party Buyer-Supplier Model. In this model, certifiers provide assurance for buyers to rely on components by developing criteria and processes by which components can be certified. In the interactions between buyers, suppliers, and certifiers, standards are developed by consensus, quality is built in during development while its cost is reduced, independence is maintained, and the cost of certification by accredited third parties is reduced. It appears that the framework could be used by third party certifiers in a component market dominated by the buyer-supplier model.

It appears that there are several factors influencing how effective a technique is for finding defects. These factors include dependencies between conditions in different code segments, how much the control structure of the code changes with changes either in individual data items or in

relationships between different data items, and whether a particular type of defect is more likely to be activated by particular control structures than others. Thus, it appears that the framework's effort to associate techniques with defect types is a necessary approach. However, it is also likely that more detailed analysis of the relationship between techniques and defects will be necessary before techniques can be recommended with confidence.

It is a significant challenge to a certifier to know under what circumstances a particular tool is applicable and produces trustworthy results. This requires a detailed understanding of the underlying technique and an ability to discover the nuances of its implementation in the tool. To improve confidence in testing results, tools must be validated for the domain and context in which they are to be used. Based upon experience, the tools are not at a mature state where the test results they produce have a significant level of confidence.

The need to evaluate the effectiveness of techniques is not a one-time problem for third party certifiers. They will be evaluating components for different domains and will have to apply different standards for different applications and domains. This on-going need for evaluation mechanisms could be addressed by the framework if it were implemented as part of an experimental lab where certification techniques and processes can be designed and evaluated. The prototype experimental lab developed by UL demonstrates the feasibility of this concept.

The opportunity exists for the Rome Laboratory Certification Framework to establish certification procedures and selection criteria for independent third party certifiers to use. In this role, the framework would continue to further the evolution of certification processes as industry needs and standards develop and change. The successful realization of this opportunity rests on the following assumptions:

- That a software parts supplier market can be created and will change how the software development industry operates,
- That independent certification will provide the confidence necessary for an application developer to buy a component rather than build or tailor one,
- That the Rome Laboratory Certification Framework can clearly establish the role it would play in the developing scenario for component certification, and
- That the present framework model can be enhanced to cover a broader view of the certification process.

Thus, a technical solution may not be as significant as the need for a business/market solution. Consideration should be given to focusing the framework for a buyer-supplier business model and to pursuing other modes of distribution than the traditional reuse repositories. An experimental lab such as that being established at UL would be a recommended approach to resolving the remaining technical issues. The results of efforts in the lab will over time broaden the framework's certification model and position components of the framework for incorporation into certification standards.

1 Introduction

1.1 Objective

The Air Force Rome Laboratory (RL) is developing a framework for certification of reusable software components under contract number F30602-94-C-0021 entitled Certification of Reusable Software Components (CRC). This framework is aimed at making software component certification usable, practical, cost-effective, and measurably beneficial. Under contract number F30602-95-C-0128, Underwriters Laboratories (UL) Inc. is conducting an independent review of the practical application of the framework addressing its usability for both the military and commercial domains from a third party test laboratory perspective.

The review addressed practical issues related to future implementation of the Rome Laboratory Framework for Certification of Reusable Software at an accredited test laboratory. The issues addressed included the goals and objectives of the framework, the costs and benefits of certification, the potential for standardization, and technology transfer.

The review was conducted concurrently with the development of the certification framework. Thus feedback from the independent review was made available when recommendations for practical applications could be further considered. As the framework is being refined, Underwriters Laboratories (UL) and other test laboratories can further address the findings when developing software component certification programs.

1.2 Background

Visual programming and object-oriented software engineering based on reusable software components are considered as key technologies that will permit substantial productivity gains in software engineering. A lack of well-documented, quality components has been described as an obstacle to cost-effective reuse. It is anticipated that in the future the commercial and military software markets for reusable software components will overlap. A dual-use certification solution that contributes to the availability of well-documented, high quality software components for both military and private sector use will be key to rapid deployment of software applications in both domains.

Much effort is currently being devoted to the development of class libraries and the population of software repositories with reusable software components. Many libraries currently contain software assets which are not being reused [Clo94]. The usability of these library assets must be addressed for the libraries to provide a meaningful service and for long term benefits in

productivity through component reuse and rework avoidance to be achieved.

Barriers to reuse of a library component by a software application developer result from a lack of information about a component and a skepticism that the component provides the function stated. The developer most likely will gauge the usability of the component based on the answers to the following questions:

1. Can I rapidly understand what it is supposed to do?
2. Does it do what it is supposed to do consistently?
3. Can I use it without having to rework or modify it?
4. Can I quickly link it in without glitches?
5. Is it portable?

Applying certification to reusable software components results in both verification of functionality and documentation that can be reviewed during the selection of a component. It has been offered as an approach to increasing the usability of reuse library assets.

1.3 Documents Reviewed

The documents developed under contract number F30602-94-C-0024 as of September 27, 1995 were baselined as the review versions for this contract. This was done to have a consistent starting point for the evaluation and to start the review process at a point in the framework development where it would not be either too late or too early to have an impact on the shape of the framework. Input too early in the process could be based on a lack of substantive information. Input too late might not be able to be assimilated into the framework and could be left for later revisions or a competing or complementary framework. Coordination of the scheduled work for both contracts played a key part in getting input from the review of work already done into the development process. The documents, their status and the date of the version that were included in the desk review documented in this interim report are listed in Table 1.1. The information provided in the documents was supplemented by material presented at the CRC Quarterly Project Reviews and working meetings, which UL staff attended.

Documents Reviewed		
Draft	Cost Benefit Plan for the Automated Certification Environment (ACE)	September 1994
Draft	Operational Concept Document for the Automated Certification Environment (ACE)	January 20, 1995
Draft	A Software Code Defect Model for Certification	May 15, 1995
Draft	Certification Tool Evaluations and Selections	May 30, 1995
Draft	Field Trial Procedures and Data Collection Guide	August 8, 1995

Table 1.1: Documents Reviewed During Desk Review

2 Evaluation of the Framework

The Certification Framework concept and approach were reviewed from a third party test laboratory perspective. Issues considered include the goals and objectives of the Certification Framework, its costs and benefits, its standardization potential, and technology transfer issues as they relate to the framework's potential for future implementation at an accredited test laboratory. These issues were addressed within the context of two questions: is the framework usable in both the government and commercial sectors and what are the benefits of and need for reusable software component certification.

2.1 Goals and Objectives of the Certification Framework

Whether the framework can meet its goals and objectives was considered in terms of the market forces that are driving the need for certification; the certification focus of the framework, in terms of users, standards, and assessment concerns; and the technical issues of the framework, such as its underlying assumptions and models.

Market Forces. The Rome Certification Framework was developed to address a perceived need to enhance the quality of reusable components in DOD reuse repositories. However, it appears that continuing problems with establishing and supporting these repositories will diminish their need for certification. The question then becomes whether there are any current market forces that will cause a need for software certification to develop. The first phase of a multi-phased market analysis by UL [UL95] to project growth in software areas that would impact the market for software certification has accumulated data on projected market growth that provides a basis for a limited assessment of the current market. The use of visual programming environments and construction from parts is considered to be the next revolution in computing technology. Over the next few years, it will stimulate the development of a component-based economy for software development with the potential for increasing the growth of the U.S. software industry by a factor of 10. Other expected benefits include an order of magnitude gain in the productivity of software applications developers, reducing time to market by a factor of 2, and increasing the usability and safety of delivered software. Standardization and certification are seen as critical to the success of this economy as it will provide the trust necessary to permit buy vs. build decisions. The assessment also indicates that there is a need for independent third party certification. Within the government sector, the Department of Commerce under the NIST Advanced Technology Program (ATP) and the Defense Department under the Advanced Research Projects Agency (ARPA) Technology Reinvestment Program (TRP) have initiated focused programs related to the development of this technology. All of these activities underscore the stimulation of market forces in the direction of a component-based software economy. As a result, the opportunity exists to influence the direction of this economy in support

of software certification activities.

UL's consideration of its role as an independent third party certifier in responding to these market forces to address the development of software certification activities is illustrated in Figure 2.1. According to this view, the end users, the verifiers, the developers, and the certifiers all play important parts in developing certification. Two essential ingredients for certification are the creation of a realization of the need for certification and standards and guidelines that can be used as the basis for certification.

Software reuse repositories are separate from but associated with the software parts or component development thrust. These repositories are generally collection facilities typically intended for "recycling" software from existing legacy systems and for "sharing" software among different development sites and projects within an organization or defense agency. The repositories contain not only parts but also other software assets such as documentation and test cases. On the surface, repositories are a good idea as they represent leveraging previous corporate investments. However, success of these repositories has not occurred and data indicates that although these repositories are consulted, actual reuse is not occurring very frequently. Stated barriers to success include the browsing problem (finding what you need) and the trust problem (how do I know this works for my application domain). There are some efforts

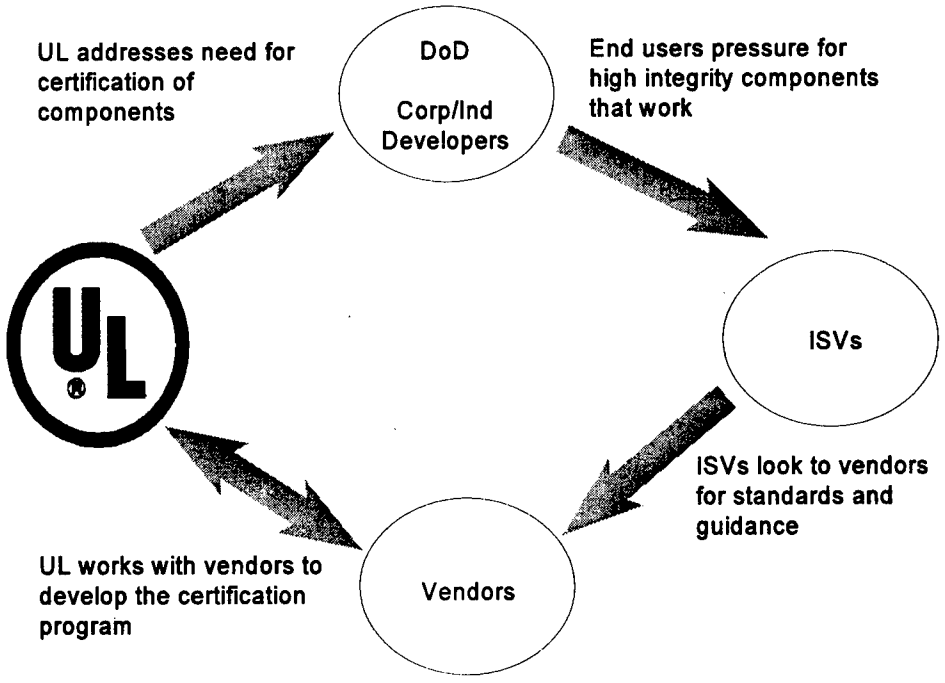

Figure 2.1: How Market Forces will Influence Software Component Certification

to oversee self-certification of the reusable asset before it is used to populate the repositories, but most of the current efforts are directed at populating repositories.

Independent third party certification could play a role in certifying existing repository components. Certification would contribute to defining the "domain of application that one can trust" which would also contribute to alleviating the browsing problem. It would also probably eliminate a lot of what is already in these repositories, forcing the software development industry to accept the fact of sunk costs. However, it is not clear that these contributions alone will make these repositories successful. Instead, it seems that the reuse repositories "placed the cart before the horse". It appears from the preliminary market data that the success of reuse repositories is linked with the growth of the component software market. The growth and stabilization of a component software market would provide the raw material for populating these repositories with independently certified components. Thus, catalysts for the effective use of repositories would include the following:

* being selective as to what is in the repository,

* packaging and certifying both new and legacy components, and

* providing management and financial incentives for reuse.

The opportunity for the Rome Certification Framework is to establish the selection criteria and procedures for independent third party certifiers to use to certify components. The success of the Certification Framework from UL's perspective rests then on three primary assertions:

1. That a software parts supplier market can be created and will change how the software development industry operates,

2. That independent certification will provide the confidence necessary for an application developer to buy a part rather than build or tailor one, and

3. That the Certification Framework can clearly establish the role it would play in the developing scenario for component certification.

Table 2.1 summarizes some of the market perspectives UL has identified that impact these assertions. If this market grows as expected, the heightened interest in certification would create a need that a well-positioned framework could immediately satisfy.

Market Resistors	Market Supporters
Job security and high wages in the software industry are tied to the current hand-crafted nature of the business. In-grained in this culture is a lack of trust for someone else's code. Indeed, many software engineers' first jobs are to maintain someone else's code when that individual has left the company and there is no documentation. Jobs where an engineer can write new code and design new systems are well sought after. Formal software testing and documentation actually may be viewed as a hassle without significant benefits.	To remain competitive in the market place, application developers will need to produce applications at a faster rate and at a lower cost than they currently do. "More features faster" is a typical requirement to maintain the competitive edge.
It will take a while for the appropriate scoping of behavior of a component to be established such that the component will be reused and there will not be a need for modification. Specifically, the parallel to the development of electronic components such as transistors and diodes may not occur because software is so easily changeable. Thus, whatever the scope of a component, the software engineer may want to and need to change it.	Through the appropriate management direction and incentives, software engineers can be motivated to use purchased components. To some extent they are already doing this by using file management utilities and mathematical and statistical routines. In certain applications, it may be preferable to purchase a single component than to buy an entire application.
Corporations will probably build their own components rather than buy components. These components will reside in internal corporate reuse repositories. Intellectual property concerns will restrict the use of externally available components in applications that are developed for resale.	Costs of software maintenance are astronomical with data showing them to be about 60-80% of software life cycle costs. Software engineering process technologies focus on reducing these costs by getting the requirements and the design right up front. However, a portion of these costs are associated with changing business needs - the use of certified components provides for rapid changeability while maintaining usability. A lot of information technology software is developed that is not for resale. Intellectual property concerns are more manageable in this scenario.

Table 2.1: Market Perspectives Impacting Assertions for Success

Certification Focus. According to Webster's, to certify is to "declare formally to be competent, valid, true, etc.". The IEEE Standard 610.1.2-1990 defines certification as either a written guarantee, a formal demonstration or the process of confirming "that a system or component complies with its specified requirements and is acceptable for operational use". The two definitions need to be mapped or merged together in order to get a clearer understanding of what software certification is. In the software world, this means defining "complies with its specified requirements" and "acceptable for operational use". Without definitions, these requirements for certification can be very subjective. In order to have a viable certification process these two elements have to be objective. They need to be defined so that a certification effort can achieve specified objectives that will either prove or disprove that software conforms to these two key statements. Some means of verifying that requirements are met in a concise, consistent manner must be defined in order for certification to take place.

The first problem is how to define the specified requirements. It is often assumed that the Software Requirements Specification (SRS) contains all the information necessary to determine if the specified requirements have been met. Though the SRS will define what the system or component should do, the possibility is high that there are additional requirements neither referenced, implied or even mentioned in the SRS. For example, the SRS might not specify that the code be written within the ANSI standard for the selected language or whether non-ANSI language extensions can be employed in the system. If extensions to the language are used, the code may not be portable to another platform that does not support them. Were guidelines or standards for reusable software followed? Were domain specific standards, such as the requirements for electronic exchange in health care environments, adhered to? The list for standards and requirements that may be extraneous to those specified in the SRS is lengthy and will continue to grow as pressures on the software industry to reduce development time forces it to focus on quality, interoperability and sharing resources.

Since operational requirements will be different from one environment to the next, reusable software may not be portable software. To ensure acceptability for operational use, reusable software may have to exist in different versions for different environments. In that case, each version will have to be certified to the requirements for its environment.

While covering a broad spectrum, software certification has to be objective to be of value to the user of the certified software. Certification needs to be repeatable and consistent: any two certifying agencies should achieve the same results when applying the same certification standard to the same software component.

A certification framework can define how to selectively apply requirements to each software asset. A synergistic relationship between the framework and software certification standards is

necessary to accomplish this goal. In specifying its requirements, the framework needs to reference existing standards that have been developed for different concerns, applications, and domains. Thus, the requirements for a particular asset would be tied to standards that apply to it. If there are key areas of the framework for which standards have not yet been defined, then the framework can identify where further research is needed and can establish requirements that would essentially be default standards. These default standards would be the starting point for new standards and the framework would be updated and revised as consensus emerged on the new standards.

The Certification Framework is beginning to provide means to measure the two components of certification. It currently focuses on the "complies with its specified requirements" portion, placing an emphasis on an implied requirement of "correctness". According to the Field Trial Procedures, "Correctness is the degree that the software is free of defects in its original context." This element of the certification framework is based on an underlying code defect model, which specifies techniques to use to find specific types of defects. The Code Defect Model, combined with the Tool Evaluations and Selections guidelines, provides a means of finding code defects. If software code is subjected to the specified procedures for using the techniques without a defect being detected, then defects of that type are, most likely, not present in the code and the code is therefore "correct" with respect to those defect types. This makes determining if software is correct an objective task.

It is not quite clear how the various aspects of the Certification Framework map to the potential set of users; in other words, who should perform which activities and at what point in the development/certification/reuse cycle. The framework includes techniques that are usually employed during the verification and validation (V&V) phases of development. While V&V is concerned with conducting activities that are effective at verifying that software meets its requirements and that its requirements are valid for its intended application, certification is more usually concerned with imposing standards on development processes, the asset itself, and the information provided to the certifier. Thus, certification would specify standards for activities that would be conducted during V&V. Repeating those activities during certification appears unnecessary. The government reuse libraries' attempts to retest assets have proven to be time consuming. The majority of code assets are only certified to a level indicating that they exist and can be successfully compiled. According to the Operational Concept Document, "A formal assessment of reusability or other quality concerns is not part of most of the repositories' asset acquisition or evaluation processes". It was found that formal assessment uses time and resources that the libraries can not afford while keeping up with the incoming flow of assets. As a result, a conclusion of the Operational Concept Document is that "There was no indication at all that more rigorous analysis or testing activities would be adopted; in fact, some repositories are planning to do less evaluation than is currently performed". Thus, it is important that guidelines

be developed to specify under what conditions a certifier would either perform the V&V activities defined in the framework or confirm that they were performed by the developer.

The framework also assumes that a user will consider code to be reusable even if it requires rework to be reused in a different application or domain from that for which it was originally developed. However, reusable assets in the commercial sector are not reworked. Attributes, such as color, size and font, are definable by the current user (application) but the inner workings are not presented for modification. Thus, the Certification Framework's selection of techniques based on rework avoidance is not applicable to a commercial asset. This discrepancy between the two views of reuse can be a major obstacle to applying the framework to certify commercial assets. If the framework defines minimum criteria for reusable assets, any asset that cannot meet those criteria would be discarded or certified under a process for single use software. In summary, the UL view of certification, illustrated in Figure 2.2, has a broader contextual focus

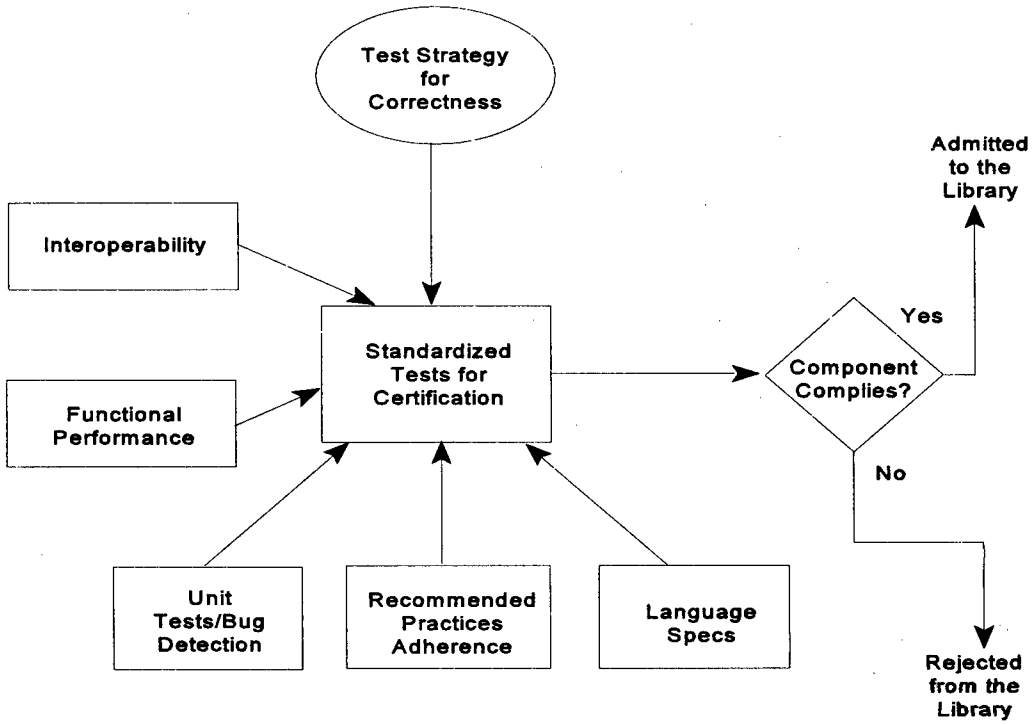

Figure 2.2: A Third Party Certifier View of Certification

but a more narrowly-defined scope than the Rome Certification Framework. The Certification Framework contains a Test Strategy for Correctness that addresses Unit Tests/Bug Detection and

parts of Functional Performance, but does not address Interoperability, Recommended Practice Adherence, and Language Specifications. Further, the scope of the "tests for certification" in the Rome framework is broader in that it provides the possibility for the certifying agency to perform all of the testing.

Technical Issues. A framework is a formal model that abstracts the central features of a family of applications which can be adapted or extended to fit the needs of particular projects [Gar]. In the case of a certification framework, it would be a model which lays out a basis for certification that can be applied to many types of repository assets. The basic model would be the same for each asset type, but there would be different selections to be made for different asset types. This is the approach taken by the Certification Framework. The underlying model on which the framework is based is that an asset is subject to defects that affect its suitability for reuse, that the defects can be categorized into classes with certain distributions and densities, and that the effectiveness of specific techniques for detecting those classes of defects can be determined. The model for software code assets is called the Software Code Defect Model, and is developed down to the level of default certification procedures by the framework. This model helps to identify where problem areas are likely to be in a code asset and, therefore, where the certification effort should focus. The framework also defines a cost/benefit model which attempts to balance certification effectiveness against cost in accordance with the notion of rework avoidance. That is, the cost of conducting certification to a particular level of defect detection is balanced against the expected cost of fixing any defects not detected in certification but that show up in the new application in which the asset is reused. Thus, a certifying agency can use the cost/benefit model to determine to what extent to carry out the certification effort. A certifier can make the stopping criteria the cost of the certification effort or the benefit of assuring a higher quality asset.

The Code Defect Model developed for the certification framework was derived from previous research in software defects and detection methods. The model was also used during the evaluation and selection of tools to be used in an automated certification environment. This model, using other sources of data, could be applied to other types of assets and tools associated with evaluating those assets, and could be extended to include concerns other than correctness. Additional research may be needed to correlate technique effectiveness with these concerns, but the basic foundation is there. The Code Defect Model could also be expanded to address the impact of language differences and other sources of variation in defect type distribution such as differences in developers, application function, development testers, and variations in methods of computing lines of code.

The concepts of the Certification Framework were incorporated into a prototype tool (the Insight Tool) that used the default data set developed in the Code Defect Model as a starting point for a certifying body. Over time, the certifying agency would collect their own data and tailor the

certification process to better meet their needs. However, the default process defined in the "Field Trial Procedures and Data Collection Guide" does not match the process defined by the default data set. As can be seen from the comparison of the process steps in the prototype, which were derived from the default data set, with those in the default process in Table 2.2, techniques have been reordered and combined in the procedures.

Insight Tool Prototype	Default Process Procedures
Code Review	Asset Readiness
Error Anomaly	Static Analysis
Structure Analysis	Code Inspection
Functional Testing	Structural-Functional Testing
Structural Testing	

Table 2.2: Comparison of Process Steps

There is no documentation as to why the changes were made, whether for example to make the process easier to understand or more consistent with typical tool usage, but the data supporting these changes does not exist or was not reported. It is therefore likely that the default process is not as effective as the framework defect model predicts since the process is not consistent with the model.

Also, it was not clear, particularly in light of the differences between the process derived from the model and that documented for the Field Trial Procedures, just what the algorithm is for selecting and ordering techniques and how it computes the effectiveness of combined techniques. One difficulty in computing the effectiveness of combined techniques is that reliance on a predicted maximum effectiveness is susceptible to error due to overlap in the error finding abilities of the combined techniques. There should also be more explicit information on how to link or disconnect between types of defects and how methods are selected. A certifying agency will encounter difficulties in building a defect profile because the certifier will be dealing with assets from many developers and may not have sufficient knowledge about defect type distributions and/or the sources of variation in the distributions. Standard profiles by component type and data on processes would need to be submitted.

While the Certification Framework is rooted in sound technical research, it is recommended that the program revisit the initial multi-level certification process [RTI93], illustrated in Figure 2.3.

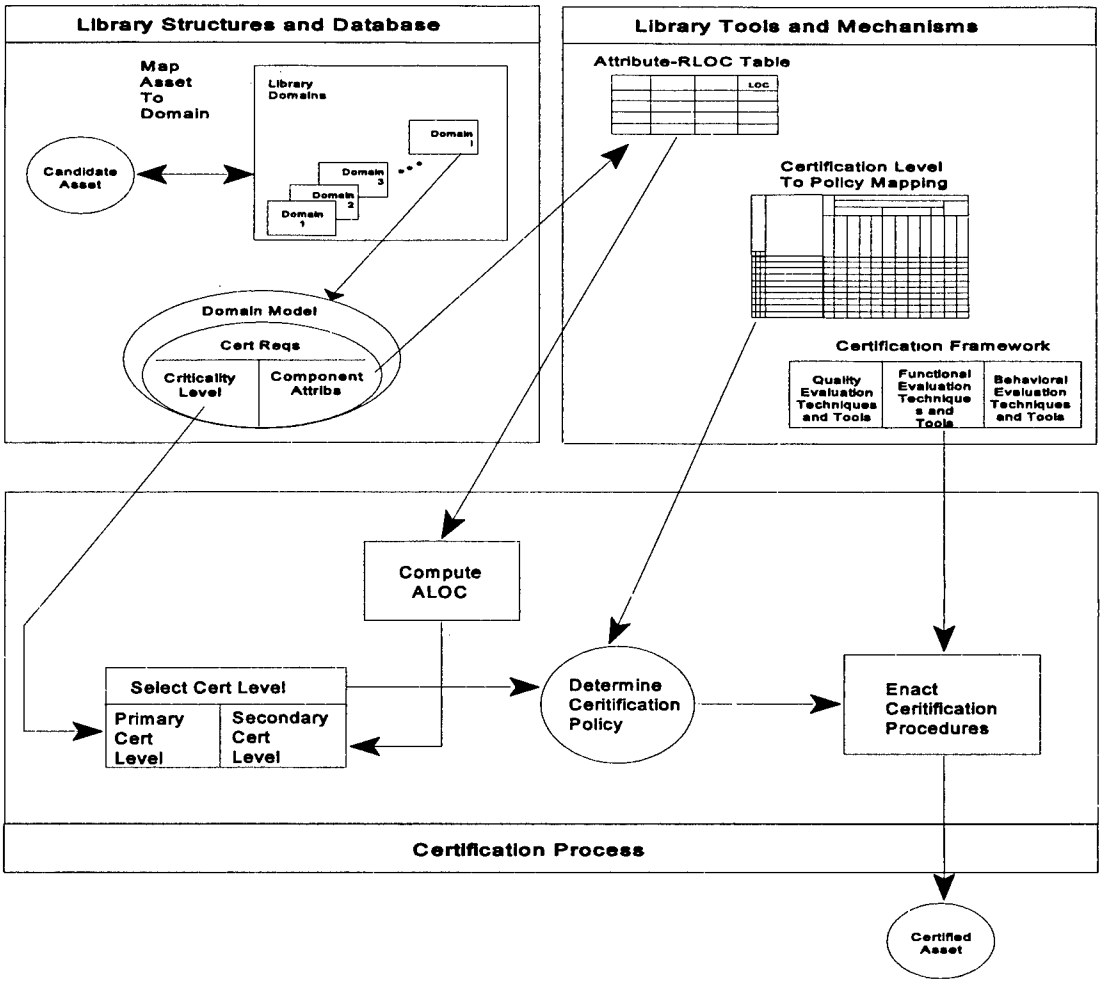

Figure 2.3: The Multi-Level Certification Process

In this initial process, a set of requirements is identified for the domain of applicability for which an asset is being developed and the asset is certified against those requirements. It is important that the domain of applicability be carefully chosen for each of the assets since the differences in requirements for different applications may result in different certification requirements. However, it is recommended that the mechanism proposed in the initial process for selecting an appropriate certification level be amended to incorporate the use of applicable standards. Standards would address process qualification, domain considerations, and criticality by product type and/or software functionality. The determination of applicable standards would make the selection mechanism objective and readily apparent. Another important aspect of the initial

process was the domain model that provided an information model of characteristic systems in a domain. This model thus specifies what the library (or certifying agency) will address for that domain. Finally, the initial process also incorporated the Rome Laboratory quality metric framework, the Software Quality Framework (SQF). This makes possible the granting of "certification credits" for the use of elements of the SQF during development or the use of elements of the SQF as requirements for certification or acceptance into the reuse library.

Finally, information requirements for software component verification are a needed component for the Certification Framework. While the equivalent of a DID needs to be developed that specifies what information needs to be provided with a component, some important pieces of information are listed below:

- part information
  - run-time image/files associated with the part
  - part category information
  - functional description or OOA/OOD attribute specification (if available)
  - behavioral specification (desirable)
    e.g. memory requirements, finite state machine descriptions, pre- and post-conditions
  - instructions for use
  - specification of key uses, pervasive use, and restrictions on use
  - interface specifications

- run-time environment information
  - classes/class libraries
  - parts/applications
  - usage specification

- development process information
  - design approach used
  - vendor test activities and summary of vendor test results

2.2 Cost and Benefits

Ultimately, the cost and benefits of the Certification Framework will be determined by the business model in which it is implemented. The framework currently appears to be addressing a business model that is based on the traditional defense IV&V model and supports extensive re-engineering. Thus, heavy certification costs are presumed to be balanced by reduced development and re-engineering costs downstream. While the benefits of certification can

perhaps be demonstrated to outweigh its costs under the current model, a greater cost/benefit could be more immediately evident under other models, such as buyer-supplier models. The distinguishing characteristics of the Third Party Buyer-Supplier Model are

- Standards are developed by consensus
- Quality is built in during development and costs are reduced
- Independence is maintained
- Cost of third party certification is reduced
- Third parties are accredited.

The traditional defense IV&V model will be increasingly replaced in the future by Buyer-Supplier models. Currently, DOD is moving to using external standards in procurement and ISO 9000 compliance is becoming more important. In the Buyer-Supplier model, software products are developed to meet consensus standards. The certifier in this model performs the role of acceptance testing, confirming that the requisite process capability is present and that standards for design and testing have been met. Although this certification may include running the product through a test suite, it is not intended to replace or repeat development testing.

While some technical research topics in certification still exist, it is not primarily a technical solution that is called for, but a business/market solution. Therefore, it is recommended that consideration be given to refocusing the framework for a buyer-supplier business model and that it target other modes of component distribution than the traditional reuse repositories.

2.3 Standardization

Standardization is a key market development activity. Table 2.3 describes some currently developing certification standards for software components. Some areas that the standards are addressing are how a part or component fits into an application environment, how a part or component talks to other components that may be local or remote, and how components talk to other components across a network. The Certification Framework has the potential to drive standardization. However, to do this there has to be consistency in application (both within a laboratory and across laboratories) of test processes determined by the framework. Here, flexibility will need to be balanced by consistency; but without consistency, certification is not possible. There is also a need for requirements on what can be used to populate a library. In driving toward standardization, the framework needs to migrate to a scheme where the developer

of the reusable component does the testing, not the library (or certifying agency). The librarian or certifier should be doing something equivalent to final acceptance testing and verifying adherence to standards. Other standardization issues include automated data collection to support refinement of the decision support element of the framework, automated test report generation, assignment of unique identifiers to reusable components, and minimizing the recertification policy when the software component has changed.

Software Component Certification

Certification Standards

	OLE	SOM/DSOM	OpenDoc	CORBA
Packaging				
Architectural Syntax	X	X		X
Component Access (Dev)	X	X		X
Naming Conventions	?	?		X
Messaging/Notification	X	X	X	X
Object Versioning	X			
Functional Performance				
Register with Environment	X			
Performs to Specification				
Handles Exceptions	X			
Communicates with Other Components	X		X	X
Shares/Uses Data of Other Components	X		X	X
No Functional Degradation with Other Components				

Table 2.3: Developing Standards for Software Component Certification

2.4 Technology Transfer

Many methodologies based on research are not in practice today because they are too difficult and time consuming to use or they do not blend with corporate culture and deadline/cost driven development. Even when methodologies are promising and practical, inadequate attention to transferring the research results into practice results in less promising methodologies becoming the defacto standard.

A key goal of the Certification Framework was to transfer software verification and validation technology from Rome Laboratory into DOD reuse repositories. The national and international move to standardization and the commercial software components market provide other opportunities for transferring this technology. An immediate transfer is the use of the Certification Framework to provide information which can help UL clients learn about approaches they can use to meet UL 1998 requirements. A more long-term transfer is the use of the framework by UL in an experimental certification lab, which is described in Section 5.1.

3 Desk Review of Procedures for Applying the Framework

3.1 Default Process

The default certification process outlined in the Field Trial Procedures differs from the process presented in the Insight Tool prototype, although both processes were supposed to be based on the same code defect model. The code defect model compiled previous research efforts to determine the dominant types of errors and the effectiveness of several techniques for finding each type. A subset of these techniques is supposed to be selected and included in the certification process based on an algorithm that compares the predicted defect distributions and the effectiveness and cost of applying the techniques with the predicted cost of rework if the defects are not corrected before the asset is reused. If the same algorithm and models were employed in creating the default process as was implemented in the Insight tool, one would expect the two processes to be identical. However, they are not: techniques have been reordered and combined in the procedures of the default process. We assume that the default process outlined in the Field Trial Procedures was modified to take into account the current certification processes in the reuse libraries and the resources they have available since the resulting process is not all that different from current reuse library processes.

This modification to the process could indicate that the selection algorithm needs to take into account additional parameters. As it stands, however, there is an unexplained discrepancy between the documented process and the process that was expected from reviewing the code defect model and the cost/benefit model. This discrepancy undermines the credibility of the process: since it is not rigorously derived from the research results and model, how can the effectiveness of the process be assured? If this divergence is the result of an attempt to take into account external factors such as how the in-place process actually works, how difficult it is to get that process modified and other disruptions in that process, then the reasons for the decisions, the effects those decisions had on the procedures, and an evaluation of the effect on the overall effectiveness of the process would require documentation.

The default process is also based on the tool selection research. This research evaluated automated tools to see which ones supported the various techniques and to what extent. Tool selection also took into account effectiveness and cost/benefit factors in selecting a default tool set for certification, but also included some tools because they are currently being used at the reuse repositories.

Although each step of the process takes an increasingly deeper look at the code, there will probably be many cases where all of the steps cannot be completed. One reason for this is that the readiness step does not require that all necessary information and components of the asset be

available for the asset to be declared ready for use or certification. Components of the asset are code modules, specification requirements, functional description, test cases and other items that would be used in the certification effort. Inputs for code inspection (step 3) require the functional description: if this wasn't provided with the asset and certification requires that a certifier write a functional description based on the code as presented, the code and the specification will always be in sync and the existence of certain types of functional defects will not be recognized.

For the process to add value to the asset, it should work like a sieve. All information and components that will be needed for all steps should be required with the asset. If they are not provided, then the asset should not be carried through the certification process and should probably be discarded from the library. Each successive step will be more selective in the inputs needed to determine adherence and the number of asset components that filter down to the succeeding steps will be fewer.

The default process also does not address regression. Correcting defects at every step is encouraged, but going back to the beginning of the certification process is not: only the current step is repeated. A fix in the code inspection step may make some code unreachable, for example, but since unreachable code was checked in the previous step, the new unreachable code would not be detected. In other words, the defect "fix" may have created one or more new defects that may not be detectable in the remaining or current steps.

Since the default tool set is used throughout the procedures, the procedures should include step-by-step instructions for using the tools to accomplish each step. This way the effort data will reflect the actual effort to complete the step and not include the tool learning curve.

3.2 Asset Readiness

The procedure for asset readiness, in the context presented, is good. However, since the tools to be used were outlined earlier in the procedures, step-by-step instructions for each bullet item would be more beneficial to the certifier.

Inputs for this step should require everything needed to complete this step and **all subsequent steps**. Exit criteria should include "all inputs have been met", including those not used in this step.

3.3 Static Analysis

Again, input should include all components necessary for this step and each subsequent step and any output from previous steps that is to be used here (whether hard or electronic copy).

One objective of this step is "Demonstration of the degree of compliance with the SPC style and quality guidelines". When major and minor defects were defined earlier in the procedures, an example of a minor defect was defined as "non-conformance to a style guideline would be a minor defect." The objective of this step is thus to remove major and minor defects; however, earlier instructions indicated that "Successful completion means no major defects are found . . .". This contradiction must be resolved either by requiring **all defects** to be fixed as part of the exit criteria for this step or by redefining a minor defect. The certifier should not be allowed to continue until **all** of the objectives of the step have been met.

3.4 Code Inspection

The procedures for this step are very comprehensive, step-by-step. Of course this can be attributed to code inspection being a purely manual step and therefore requiring more guidance to achieve the desired outcome. The code checklist is very comprehensive. Because of the comprehensive nature of the checklist, the best results would be achieved if two or more inspectors reviewed the code, which is the case for most code reviews. With more than one inspector, errors that may be detected by one inspector and not the others are caught. Multiple inspectors also alleviate the problem of an inspector seeing what he wants to see after looking at code for some length of time. Even a programmer presented with the exact line of code where a defect exists might overlook the defect because she has spent so much time looking at the code that the defects have become invisible. Often, even after hours of trying to fix a bug, the error may still not be apparent. A fresh set of eyes can look at the code, however, and within a few minutes locate and fix the problem. This is why methodologies that incorporate code reviews and inspections in the software life cycle use teams of programmers and analysts to review and inspect code.

The objective of "Completeness - assessment of the adequacy of the functional description" appears to be a mis-statement of the problem. The functional description of the asset should have been written before the code and the asset design and the code that implements it should have been written in accordance with all elements of the functional description. If there is a discrepancy between the code and the functional description, the problem lies either in the design of the asset or the code itself. The functional description of the asset should be considered the blue print for the asset. All other components of the asset must be based upon the functional description. The assessment should be "of the adequacy of the code to meet the functional description".

3.5 Hybrid Structural-Functional Testing

Achieving the objectives of this step is the primary means by which this certification process will

add value to an asset. The objectives of determining whether an asset "performs its intended function within the specified requirements" or "is complete with respect to the functional specification or description" are often left by the development team for the user to determine. It is frequently the case that by the time software is ready for this type of testing, the budget has been spent (or overspent) and production is far behind schedule. Thus, this type of testing is usually done by the user in a production environment. To have this testing completed up front by an independent certification body would add a great deal of confidence to the user. Of course, this type of testing is also resource and time prohibitive, and most assets are not certified to this level. The automated tools help reduce the cost of testing, but the certifier has to come to a thorough understanding of the functional description and requirements to develop appropriate test cases, which in itself can be a time consuming process.

4 Application of the Framework

The goal of this portion of the project was to evaluate the framework by applying it. To that end, an asset was selected and the default certification process documented in the Field Trial Procedures and Data Collection Guide was applied to it. A scenario was developed that incorporated domain information, application system requirements, and development process summaries and analyses. This scenario provided the certification context from which certification concerns and criteria were identified. Based on these concerns and criteria, certification requirements for the component were established.

A reusable Ada software component was selected from the Air Force DSRS (Defense Software Reuse System) reuse library that met the functional needs of the scenario. This component is a heap sort routine that was commercially produced as a reusable component and is part of a set of reusable components that is available for purchase. Thus, it could be used in many different domains and for many different types of applications. Benchmark components were created for the routine, including a functional description of a heap sort, specifications and requirements (from the scenario), instrumented source code, control & data flow models, path analysis models, test cases, and fault sets. We also created a C version of the heap sort and its associated benchmark components.

4.1 Scenario

To represent the extreme end of criticality and complexity that would have to be addressed by the certification framework, a case study from an earlier Rome Laboratory effort [Sch91] that focused on design and evaluation methods for highly dependable, complex space applications in a BMC^3 domain was selected. This case study was adapted into a scenario that assumes both a component-based engineering process where systems are developed by designing architectures, selecting components in accordance with the requirements of the architectures, and building "from scratch" only that software that is necessary to integrate the components and a requirement that all components be certified. In this scenario, the certification process needs to address the following concerns:

Function	What it does
Correctness	If it correctly implements its specifications and meets its requirements
Range of Applicability	Its assumptions about inputs, usage, and domain

Interoperability	The types of components it can work with and the types of architectures for which it is suited
Standards	The standards it meets
Selection Criteria	What are the important factors in selecting it for use in a particular application

The development context for the scenario is an iterative design process broadly divided into three phases: baseline determination, initial design, and design refinement. The baseline determination phase determines resource requirements and allocation for the basic architectural and algorithmic structures of the system. The initial design phase consists of trade-off studies to select from among the design options being considered the one(s) that meet the system requirements. It also identifies any common functions that are candidates for implementation using certified components. The design refinement phase explores the selected design option(s) to discover and remove any deficiencies in concepts or requirements.

The system in this scenario manages the assignment and use of space-based weapons for multiple hostile boosters. The top-level view of the system architecture is illustrated in Figure 4.1.

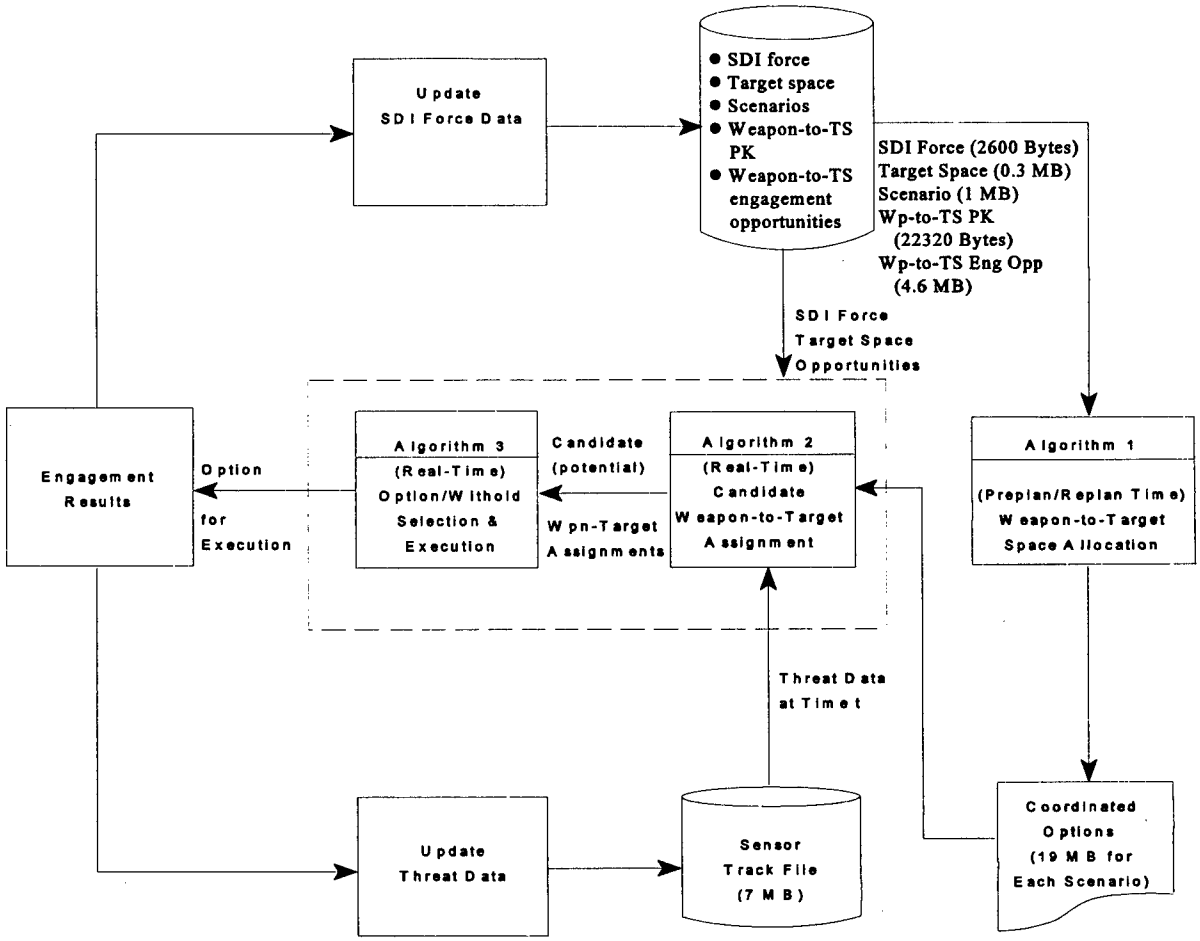

Figure 4.1: Top-Level View of System Architecture for Scenario [Sch91]

In this scenario it is assumed that during the baseline and initial design phases, a high-level design of a Weapon to Target Assignment and Target Sequencing (WTA/TS) Algorithm, Algorithm 2 in Figure 4.1, was created. This algorithm clusters targets and assigns and sequences weapons to the target clusters. The WTA/TS algorithm was in turn partitioned into four functional components: target cluster definition (TCD), weapon-to-target cluster allocation (WTC), weapon-to-target assignment (WA), and target-sequencing (TS). These functions were decomposed into subfunctions down to the level of identifiable matrix, sorting, linear

programming, and integer programming operations utilized by the algorithm.

Figure 4.2 depicts a model of the WTA/TS algorithm which depicts how the algorithm is decomposed and where the reusable components are used. The multiple dimensions of this

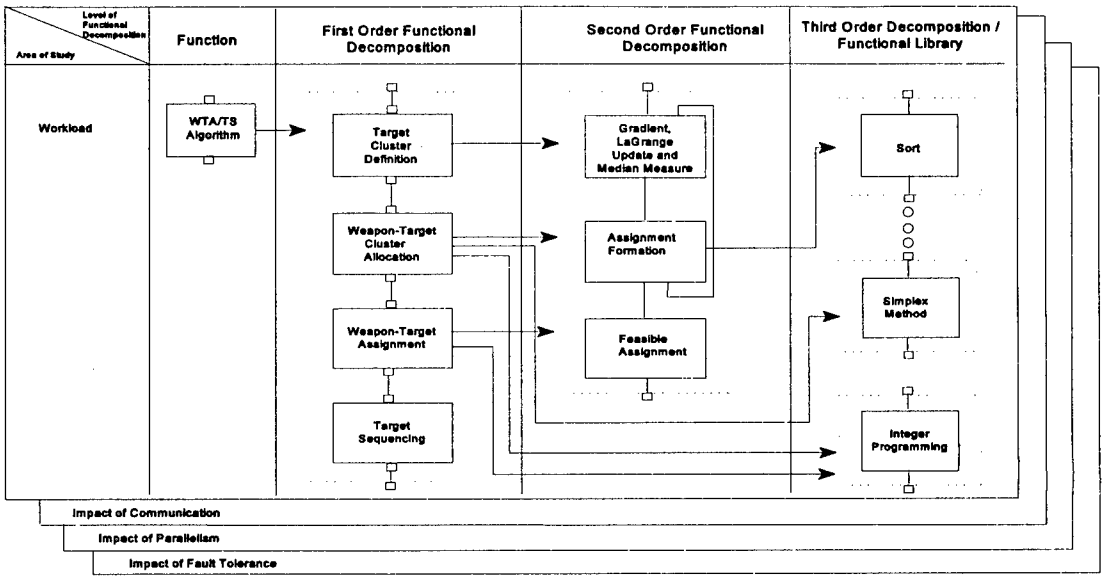

Figure 4.2: WTA/TS Algorithm Structure [Sch91]

diagram reflect the need to be able to represent different aspects of system behavior, such as functional decomposition, communication, parallelism, and fault tolerance. In this figure, the top-level functional components are shown. Three of these components, Target Cluster Definition (TCD), Weapon-to-Target Cluster Allocation (WTC), and Weapon-to-Target Assignment (WA), share in their respective subgraphs the three component functions shown in the next level of functional decomposition. These three components are the Gradient, LaGrange Update and Median Measure (GLM); the Assignment Formation; and the Feasible Assignment. The lowest level of functional decomposition consists of the common functions, which are shared across multiple levels of decomposition. Shown at this level are sort, simplex method, and integer programming modules. These common subfunctions were selected as potential candidates for implementation using reusable components.

A performance analysis of the algorithm conducted during the original case study determined that, in terms of workload, TCD is the dominant top-level component. Another performance analysis determined that an important factor in improving performance was a revision in the design of the sort function to use an NlogN algorithm rather than an N Squared algorithm. This makes this common subfunction a high-priority reuse candidate.

Although only the heap sort routine was evaluated in this effort, the scenario presents the opportunity to investigate the application of the framework to three "levels" of components: an atomic component (the heap sort routine), a component composed of other components (the WTA/TS algorithm), and an architecture for building a system using components (the space-based BMC^3 system).

4.2 Certification Requirements

According to the guidelines of the Rome Laboratory framework, specific requirements concerns, called quality factors, are identified and used to select the appropriate certification scope and confidence levels. The types of defects associated with these concerns determine the scope and the "level" of the concern determines the confidence level. As noted in the description of the scenario, function, correctness, range of applicability, interoperability, standards, and selection criteria are concerns that need to be addressed. Of the quality factors specified by the framework, criticality (component and system level), fault tolerance, performance, reliability, and safety are all applicable and are all applicable at their highest individual levels. For each of these, we identified the associated requirement(s), the framework certification level that addresses it, and the step in the default process that would evaluate whether or not the component met that requirement. The types of concerns and the level of their individual requirements require that the highest confidence levels of the framework be attained; i.e., that the full range of evaluation techniques be applied within each scope level. However, the default certification process was targeted to a less demanding scenario and as a result it does not include the full range of techniques. Table 4.1 summarizes the concerns, the requirement(s) associated with each concern, the relevant framework scope level, and the relevant step(s) in the default process. As noted in the table, the framework does not address the standards and selection criteria factors. Since the default certification process only addressed that part of the framework concerned with latent defects, function and correctness are the only scenario concerns that the default process evaluates the component against.

Concern	Requirement	Framework Scope Level	Default Process Step
Function	Performs Heap Sort	Latent: correctness	Step 4: Testing
Correctness: Implements Specifications	No coding defects	Latent: correctness & completeness	Step 1: Readiness Step 2: Static Analysis Step 3: Code Inspection Step 4: Testing
Correctness: Meets Performance Requirements	$N\log_2 N$; .05sec/target for complete WTA/TS	Operational Robustness	NA
Correctness: Meets Reliability Requirements	High	Operational	NA
Range of Applicability	Can sort targets, weapon/target pairs, clusters	Robustness	NA
Interoperability	Ada; Real-Time OS	Interoperability	NA
Standards	None	NA	NA
Selection Criteria	Efficiency & Versatility; Data Types	NA	NA
Component Criticality	Severe operational limits	All	NA
System Criticality	Life-threatening hazards	All	NA
Fault Tolerance	Provide service in spite of faults	All	NA
Performance	Real-Time, High Workload	All	NA
Reliability	10^{-9}	All	NA
Safety	Unrecoverable environmental damage; many people killed	All	NA

Table 4.1: Certification Requirements for the Selected Component

4.3 Certification Process

The default certification procedures that were followed in this application of the framework are illustrated in Figure 4.3. The complete default certification process was applied to the Ada version of the component; the testing portion of the process was applied to the C version.

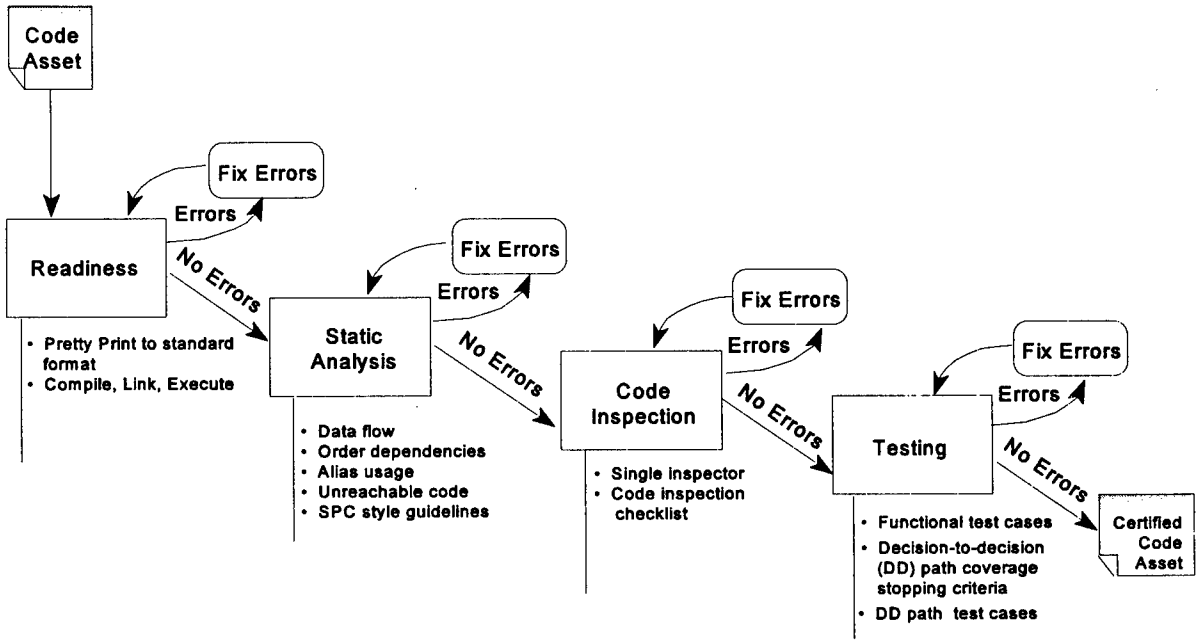

Figure 4.3: Overview of Default Certification Process

Tools to implement the techniques specified by the procedures were selected based on the Certification Tool Evaluations and Selections document. The selected tools are summarized in Table 4.2.

Benchmark components were created to provide a reference basis for evaluating the techniques and tools. These components consisted of instrumented code; models of the control flow, data flow, and path structure of the code; and sets of faults to be injected in the code before applying the techniques. The execution coverage of the code and any analysis results produced by the tools were compared to the benchmarks to identify any discrepancies between actual results and

expected results.

Tool	Process Step
Ada Component	
Apex Ada	Step 1: Readiness
AdaQuest	Step 2: Static Analysis
AdaWise	Step 2: Static Analysis
TestMate	Step 4: Testing
C Component	
Sun C	Step 4: Testing
SoftwareTestWorks	Step 4: Testing

Table 4.2: Tools Used in the Application of the Default Certification Process

The application was carried out by a team of people representing a wide range of experience. The testing team leader has more than 10 years experience in software evaluation and testing and was in charge of coordinating the activities of the other staff in setting up the tools and executing the procedures. The team leader also participated in the design of test cases and the development of drivers to execute the test cases. Since the CRC survey of reuse libraries had indicated that a number of certification activities were performed by people who were not experienced developers or testers, the testing team included a tester who has no software development or testing experience but who has more than 10 years experience with computerized processes (the use of computer-based tools to execute the procedures of a well-defined, structured process). This tester executed all of the default process procedures for the Ada version of the component, created test cases and drivers, managed the configuration of the artifacts associated with the procedures, and collated the results of the tests and analyses. To cover the type of certifier who is a more experienced tester, a second tester was included on the team who had about 3 years experience with software development and testing, including a working knowledge of the C language. This tester developed and executed the test cases for the C version of the component. The testing team also included two additional people in a support capacity. One support person

was in charge of tool installation and integration. Because there was very little experience with Ada among the other members of the team, a second support person with considerable Ada experience, both in Ada programming and in the development and evaluation of Ada compilers, was included as a resource to answer questions and troubleshoot problems. The experience level of the testing team is summarized in Table 4.3.

Testing Team Staff	Level of Experience by Category					
	Ada	C	Software Development	Testing	Computerized Processes	Tool Support
Testing Team Leader	low	high	high	high	high	low
Ada Tester	none	none	none	none	high	high
C Tester	none	medium	medium	medium	high	none
Lab support person	none	medium	high	high	high	high
Ada resource person	high	medium	high	high	high	none

Table 4.3: Testing Team Experience

4.4 Application Results

Table 4.4 presents a summary of the results of applying the default certification process to the Ada component. The key finding from this activity is that the application of the procedures by a third party certifier on a component designed and produced for the commercial market may not be an appropriate application for the framework. It would be more practical for certifiers to use the framework to derive requirements for testing and evaluation of components that the developers would implement. The certifiers would then verify that the developers had implemented the specified activities.

The activities specified for the testing step were very labor-intensive, and in the case of a well-developed and verified component, repetitive. The defects that were found by the other steps of the process were mostly violations of programming style, which could in some cases lead to other types of defects that affect the functionality of the code. Another difficulty with applying

these procedures to components that are particularly designed for reuse is that the generic nature of the component increases the scope of the testing. For example, the Ada heap sort component was not defined for any specific data type or sorting order, but would "instantiate" itself according to the data type of the items to be sorted and the ordering function passed to it. This is equivalent to testing not only as many programs as there are valid data types that can be constructed in Ada and functions that can be created to define sort orders, but also that the Ada structures that make the code "generic" are correctly implemented.

According to the default process, code can not progress to a higher certification level if it is found to have a major defect. The framework does not sufficiently differentiate between major and minor defects, leaving too much discretion to the certifier. The intended domain of the code should be specified in as much detail as possible to ensure more appropriate testing of the component. Otherwise, the certifier has to have domain information and scenarios in-house to "create" this information before testing can begin.

The testing of the C component highlighted the need for more detailed documentation of the design and implementation details of components to prevent their misuse. The developer of this code had used the first and last elements of the sort array structure as sentinels. This fact was not known to the tester who assumed the usual C convention of indexing an array of N elements from 0 to N-1. When only positive numbers were used in the test cases, the result was a correctly sorted array since all of the input numbers sorted higher than the 0's with which the array structure was initialized. However, when negative numbers were used in the test cases, the result was a sorted array in which the negative numbers had been replaced by 0's. Thus, although there was not really a defect in the component, the test cases "failed" and significant debugging effort was expended to identify the problem.

The application of the techniques requires a considerable investment in tools, training, and qualified staff. Although many of the automated techniques can be applied by inexperienced staff, the test data is hard to interpret. The analysis of the results to determine the significance of any detected defects requires at minimum a medium level of software engineering experience. The development of test cases, particularly in the absence of any development infrastructure, also requires at least this level of experience. The initial determination of the utility and applicability of the selected tools requires a high level of experience and a good knowledge of software testing concepts.

Based upon experience, the tools are not at a mature state where the test results they produce have a significant level of confidence. The benchmark components were very useful in evaluating the tools. From these models we knew how many statements, segments, decisions, paths, and other structural components existed in each of the modules of the component. This

provided a yardstick to compare the tool results against. When a discrepancy occurred between the benchmarks and the Ada coverage metrics computed by the tool we were using, we discovered that the version of the tool we were using did not correctly handle replicated generics. An updated version of the tool was installed and successfully used for the remainder of the testing. It is a significant challenge to a certifier, whether actually using a tool or evaluating its use by someone else, to know under what circumstances the tool is applicable and produces trustworthy results. This requires a detailed understanding of the underlying technique and an ability to discover the nuances of its implementation in the tool. To improve confidence in testing results, the tools used must be validated for the domain and context in which they are used.

Table 4.4: Summary of Results for Ada Component

	Default Certification Process Activities			
	Readiness	**Static Analysis**	**Code Inspection**	**Testing**
Level of Effort (hours)	4 hrs (including setting up views and downloading asset)	AdaQuest:1 hr AdaWise: 1 hr	Preparation: 8 hrs Inspection: 2 hrs	Steps 1-3: 16 hrs Steps 4-8:
Number of Defects Found				
Computational			*2 indeterminate**	
Data			4 minor	
Interface		2 minor	3 minor	
Logic		2 minor	*3 indeterminate**	
Other			2 minor	
Total		4 minor	9 minor	0
Problems in Applying Techniques	None	None	Some of the questions did not apply or could not be answered	Too much "infrastructure" had to be created by tester
Problems in Using Tools	✧ Installation ✧ File Structure ✧ Setting parameters, attributes, etc. ✧ Unclear error messages	✧ None with AdaQuest ✧ Installation: AdaWise		✧ Licensing ✧ Installation ✧ Accessing ASIS ✧ No test generation ✧ Missing features ✧ Inadequate documentation
Problems with Process Guidance	✧ More guidance on tool usage			✧ Guidelines for creating test cases didn't fit generic case

* *indeterminate* indicates a checklist question that could not be definitively answered

Tables 4.5 - 4.7 summarize the results of testing the C version of the heap sort component. As for the Ada version, this component consisted of a main function called heapsort and an embedded function called sift. The sift function is called at two different points by heapsort, once in a FOR loop and once in a WHILE loop. A driver was constructed to call heapsort and pass it the data for each of the test cases. The testing consisted of developing and running test cases against the original code and then injecting faults in the code and running the test cases again. Segment, decision, and path coverage metrics were computed for both the faulted and the fault-free runs of the test cases. The test cases are described in Table 4.5 and the faults that were identified by each test case are indicated.

Test Case	Description	Faults Found
1	zero value	2, 4, 5, 7, 9, 10
2	one value	2, 4, 5, 7, 9, 10
3	two values	2, 4, 5, 7, 9, 10
4	positive integers in sorted order	2, 4, 5, 7, 8, 9, 10
5	positive integers in reverse order	2, 4, 5, 7, 9, 10
6	positive integers in random order	2, 4, 5, 7, 8, 9, 10
7	positive integers and a zero value	2, 4, 5, 7, 8, 9, 10
8	negative integers in sorted order	2, 4, 5, 7, 8, 9, 10
9	negative integers in reverse order	2, 4, 5, 7, 9, 10
10	negative integers in random order	2, 4, 5, 7, 9, 10
11	negative integers and a zero value	2, 4, 5, 7, 8, 9, 10
12	positive & negative integers in sorted order	2, 4, 5, 7, 8, 9, 10
13	positive & negative integers in reverse order	2, 4. 5, 7, 9, 10
14	positive & negative integers in random order	2, 4, 5, 7, 9, 10
15	positive & negative integers and a zero value	2, 4, 5, 7, 9, 10
16	positive & negative integers with duplicates	2, 4, 5, 7, 9, 10
17	positive & negative integers with multiple zeroes	2, 4, 5, 7, 9, 10
18	path test two numbers	2, 4, 5, 7, 8, 9, 10
19	path test three numbers reverse order	2, 4, 5, 7, 9, 10
20	path test three numbers sorted order	2, 4, 5, 7, 8, 9, 10
21	path test four numbers sorted order	2, 4, 5, 7, 8, 9, 10
22	path test four numbers reverse order	2, 4, 5, 7, 9, 10

Table 4.5: Summary of Test Cases for C component

Table 4.6 summarizes the faults that were injected in the code, including the code segment number where the fault was injected and what effect the fault had on the functionality of the code component. The faults were injected by making small mutations in data variables, logical and arithmetical operators, and array subscripts. The Table also identifies the defect type associated with the fault and indicates which test case(s) detected the fault. There were four data defects, one of which was not detected by any of the test cases. The remaining three data defects were detected by all of the test cases. The two logic defects caused the program to abort with a segmentation fault; in one case, it produced invalid results for all test cases before aborting. The one computational defect was detected by all of the test cases. There were three interface defects with three different results: the first was detected by all test cases, the second was detected by none of the test cases although the program aborted with a segmentation fault, and the third was detected by slightly less than half of the test cases but resulted in correct results for the others. Interestingly, all of the test cases in which the input data was already in the prescribed sort order caused the sort to fail and the fault to be detected. There was one case where two faults were injected, a data defect and a logic defect. The data defect was the same data defect that was detected by none of the test cases when inserted singly. In this case, in combination with a logic fault, all of the test cases produced invalid results.

Fault No.	Segment No.	Description	Mutation Type	Defect Type	Test Case Found By
1	H1	Endpoint passed to Sift rather than midpoint	Change in data variable	Data	none
2	H1 S1	Endpoint passed to Sift rather than midpoint Sift processed from endpoint rather than midpoint	Change in data variable Change in data variable	Data Logic	all
3	H3	Heapsort processed beyond end of array	Change in logical operator	Logic	crashed
4	S2	Sift miscalculated endpoint	Change in arithmetical operator	Computational	all
5	M12	Wrong array size passed to Heapsort	Change in data variable	Interface	all
6	M12	Heapsort called too many times	Change in data variable	Interface	none/crashed
7	S1	Incorrect array element stored by Sift	Change in array subscript	Data	all
8	H2	Incorrect array size passed to Sift	Change in data variable	Interface	4,6,7,8,11,12, 18,20,21
9	S1	Sift processed wrong segment of array	Change in logical operator	Logic	all/crashed
10	S6	Sift swapped wrong elements	Change in array subscript	Data	all

Table 4.6: Summary of Faults Injected in C Component

Table 4.7 summarizes the path coverage results from running the test cases on both the non-faulted code and for each of the fault sets. There are several interesting observations relevant to the use of path coverage metrics as a measure of testing effectiveness for this component. First, note that there are 8 paths identified for the heapsort module of the sort component and 5 for the sift module. It was noted in the documentation for the testing tool that invalid paths were sometimes identified; that is, paths that are structurally correct but which are logically infeasible.

Code ID	Path ID	No Fault	Number of Hits Per Path Per Fault ID									
			1	2	3	4	5	6	7	8	9	10
heapsort	1	20	20	20	0	20	21	21	20	20		20
	2	0	0	0	0	0	0	0	0	0		0
	3	0	1	1	0	0	0	0	0	0		0
	4	0	0	0	0	0	0	0	0	0		0
	5	0	0	0	0	0	0	0	0	0		0
	6	0	0	0	0	0	0	0	0	0		0
	7	1	0	0	0	1	1	1	1	1		1
	8	1	1	1	0	1	0	1	1	1		1
	cov %	37.5	37.5	37.5	0	37.5	25	37.5	37.5	37.5		37.5
Sift	1	0	0	0	09	4	1	18	12	5		1
	2	17	17	13	0	13	26	117	6	10		13
	3	19	19	71	0	28	30	55	43	23		25
	4	48	48	73	0	39	64	180	23	42		45
	5	20	73	0	517	29	21	21	20	24		20
	cov %	80	80	60	20	100	100	100	100	100		100
Fault Location			H1	H1 S1	H3	S2	M12	M12	S1	H2	S1	S6
Results		✓	✓	✘	☼	✘	✘	✓/☼	✘	✓/✘	✘/☼	✘

Table 4.7: Summary of Path Coverage for C Component

For example, consider the path diagram in Figure 4.4. In this example, segments 1 through 5 of the heapsort module are indicated as nodes in a path diagram (note that segments 6 through 7 are

not shown). There is a FOR loop in segment 1 and a WHILE loop in segment 3 that cause branches. The resulting paths are identified as

(1) $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7$
(2) $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 7$
(3) $1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 7$
(4) $1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 7$
(5) $1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7$
(6) $1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 7$
(7) $1 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 7$
(8) $1 \rightarrow 3 \rightarrow 5 \rightarrow 7$

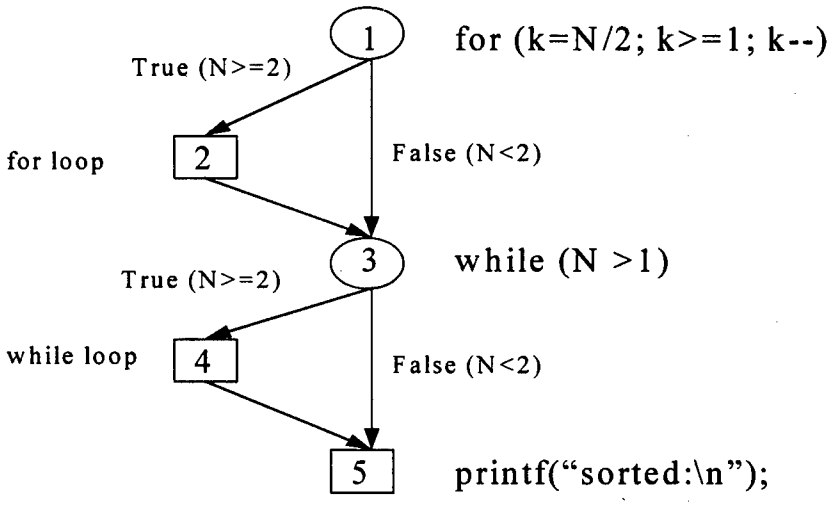

Figure 4.4: Heap Sort Paths

However, paths 3 and 4 are not logically feasible because if the condition at segment 1 is true,

then the condition at segment 3 is true. In other words, the two decision conditions are not independent, and the branch from segment 1 to segment 2 predetermines the branch from segment 3 to segment 4. Likewise, paths 5 and 6 are not logically feasible. Although the path coverage metric computed by the tool is 3 out of 8 (37.5%) for the non-faulted test run, it is actually 3 out of 3 (100%).

Looking at the entries across the Table for the different faulted runs, we see very little difference in the coverage metrics; however, the result of applying the set of test cases varies from all test cases detecting the fault to no test case detecting the fault. Also, looking at the entries for the coverage metrics for the sift module, we see that none of the test cases detected the data defect associated with fault 1 although the associated coverage metric is 80%, but all of the test cases detected the defects associated with fault 2 with a 60% coverage metric. Likewise, faults 6, 7, and 8 resulted in detection by none, all, and some of the test cases although 100% coverage was achieved in all three cases for both the heapsort and sift modules of the component.

In the case of fault 1, the data passed to sift when it is initially called by heapsort was mutated so that the value passed as the index into the array was the endpoint rather than the midpoint. The result was that heapsort called sift over and over with no activity being initiated by sift until the loop control had decremented the improperly set index until it became equal to the midpoint of the array, which was its originally intended value. At that point, the remainder of the sort was executed as expected and the array was properly sorted. Although the code was defective, the defect was not material to the functionality of the algorithm; its only effect would be a possible decrease in performance since it was wasting time. Thus, if there were test cases that tested whether or not the component met the performance specifications of a heap sort routine, those test cases might have detected this defect. Although the results of the test cases were as expected, there are clues in the coverage data that point to a problem with the code. Referring to Table 4.8, we see that this fault resulted in the heap sort paths 1, 3, and 8 being executed. However, as noted above, path 3 is an invalid path, and something would have had to break the dependence between the conditions determining the branches at segments 1 and 3 for this path to have been activated. This "something" is the mutation in the data defining the starting point for building heaps. Also, we note from Table 4.8 that one of the valid heap sort paths, path 7, was not executed in this test run and that the number of hits of the entry-exit path (path 5) of the sift module has increased significantly over the fault-free run, whereas the number of hits of all of the other sift paths remained the same.

It appears that there are several factors influencing how effective a technique is for finding defects. These factors include dependencies between conditions in different code segments, how much the control structure of the code changes with changes either in individual data items or in relationships between different data items, and whether a particular type of defect is more likely

to be activated by particular control structures than others. Thus, it appears that the framework's effort to associate techniques with defect types is a necessary approach. However, it is also likely that more detailed analysis of the relationship between techniques and defects will be necessary before techniques can be recommended with confidence.

5 Instantiation of the Framework for a Third Party Certifier

The goal of the application portion of the project was to evaluate the framework by applying it. To that end, an asset was selected and the default certification process was applied to it. However, as discussed in Section 2.1 above, there are market and technical issues that would prevent the direct implementation of the Certification Framework and its algorithm for selecting and ordering techniques in a certification environment such as UL's. Also, there is an issue of how to measure individual and combined technique effectiveness given that different certification requirements exist for different domains and that characteristics of systems in different domains affect the applicability and effectiveness of defect detection techniques. While the framework recognizes this and provides for the creation of defect/detection models particular to a domain or application, the default process is tied to a particular model and particular assumptions about the reuse context. Thus, while the application of the default procedures provides some feedback on their use, it does not evaluate the central concept of applying the framework, which is its instantiation for a particular certification scenario. Therefore, as part of the application of the framework, we investigated how the framework could be implemented by a third-party certifier.

It appears that the Rome Certification Framework could be used by third party certifiers in a component market dominated by the buyer-supplier model. Figure 5.1 illustrates this model. In this model, the certifiers provide the assurance needed for buyers to rely on components that the suppliers produce by developing criteria and processes by which components can be certified. Standards would be developed and used in this process to ensure that the requirements for certification meet the needs of both the buyers and the suppliers. In other words, the requirements would result in the specification of processes and techniques that the developers could clearly see the use and effectiveness of and that the suppliers would trust to produce quality components that meet their requirements. The Rome framework can support this process by providing a framework for selecting and evaluating appropriate and effective evaluation techniques for specific certification concerns.

Since third party certifiers will be evaluating components for different domains and will have to apply different standards for different applications and domains, the need to evaluate the effectiveness of techniques is not a one-time problem. Thus, the certifier will need many sets of tools and techniques as well as software benchmarks by which the tools and techniques can be evaluated for particular certification scenarios.

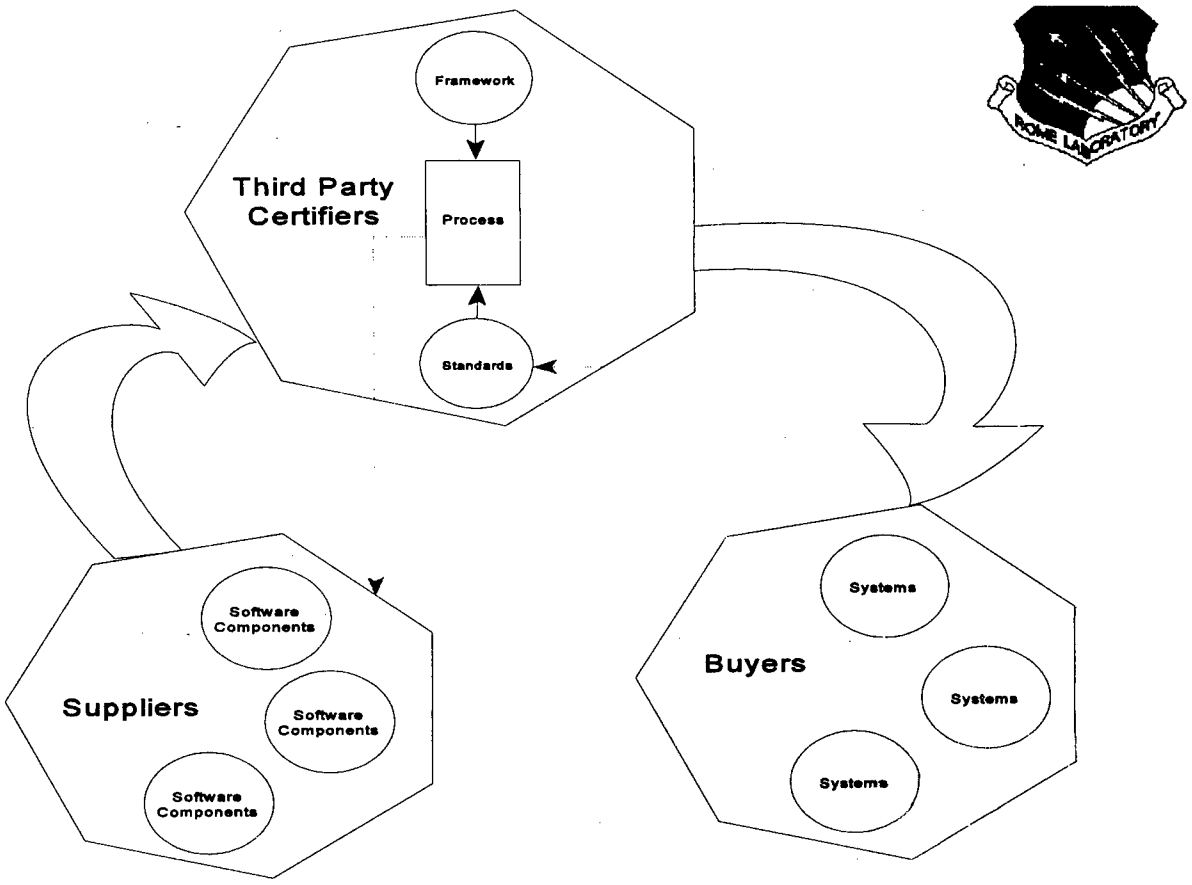

Figure 5.1: Buyer-Supplier Model

The framework could be implemented by a third-party certifier as one component of an experimental lab which would be used to evolve processes as industry needs and standards develop and change. Thus, part of our application of the framework was a proof-of-concept development of such an experimental lab. In addition to enhancing the evaluation of the framework, this experimental lab also directly positions the framework for potential use by a third party certifier.

5.1 Experimental Certification Lab

The prototype experimental certification lab demonstrates how the Rome Certification Framework can be used by third party certifiers to design certification processes and to evaluate the effectiveness of techniques proposed by the certification framework. The experimental lab was designed to meet the following requirements:

(1) support multiple languages and platforms,

(2) include automated tools for development and testing,

(3) include bench marking facilities that support the evaluation of the effectiveness of the techniques specified by the framework,

(4) link and access all of its facilities via an intranet web application, and

(5) interface with the Rome Laboratory Automated Certification Environment (ACE) prototype.

The existing UL Software Test Laboratory provided the physical components of the experimental lab. The UL lab supports safety certification at UL by providing support for internal research relevant to the evaluation of client software. As such, it includes a wide range of strategic hardware platforms and software tools. Strategic platforms are networked together to allow for resource sharing as well as information sharing (providing for shared network directories on a primary server).

The network is connected via a 10-base T Ethernet topology, utilizing a 3com LinkSwitch as a router and configurable firewall. The primary server is a Compaq Proliant 4500, running Windows NT 3.51. The machine has 128M of user memory, dual Pentium 133 MHZ processors and 16 Gigabytes of hard disks (effectively 12 Gigabytes as disk array is configured with RAID 4 hard disk recovery backup system). Disk shares are provided via NFS; exported common directories for share among various heterogeneous platforms including UNIX Solaris, MAC and O/S 2; as well as shares with Windows/Windows NT/Windows 95 clients.

The benchmarking facilities of the experimental lab provide descriptive and quantitative information about a component so that the results of applying specific defect detection techniques can be compared with an expectation of what those results should have been. Should there be any discrepancy between expected and actual results, the cause would be investigated and any deficiencies in the technique with respect to a particular application identified.

The objects that comprise a benchmark are constructed within a context defined by a scenario which provides the system-level requirements from which the component-level requirements and applicable standards are derived. At the component level, the benchmark consists of the following objects:

- a functional description
- specifications and requirements
- source code
- instrumented source code
- control & data flow models
- path analysis models
- test cases
- fault sets.

These objects can be integrated by a hierarchical model that connects the objects into views of the system-level architecture. The model shown in Figure 4.2 for the scenario developed for the application of the framework is a component of such a model. This model is a collection of models which describe several architectural perspectives of a subsystem of an application. The perspective shown in this Figure is the functional decomposition of the subsystem into individual, reusable components. This model allows the software to be described at the workload level, which enables performance, communication, and resource utilization issues to be highlighted and baselined.

The facilities of the experimental lab form an intranet web that can be accessed via a hyperlinked interface. The key facilities are access to files from individual certification projects; the ability to reference, create, and enact certification processes; and access to reference information. The overall structure of the web interface is diagramed in Figure 5.2.

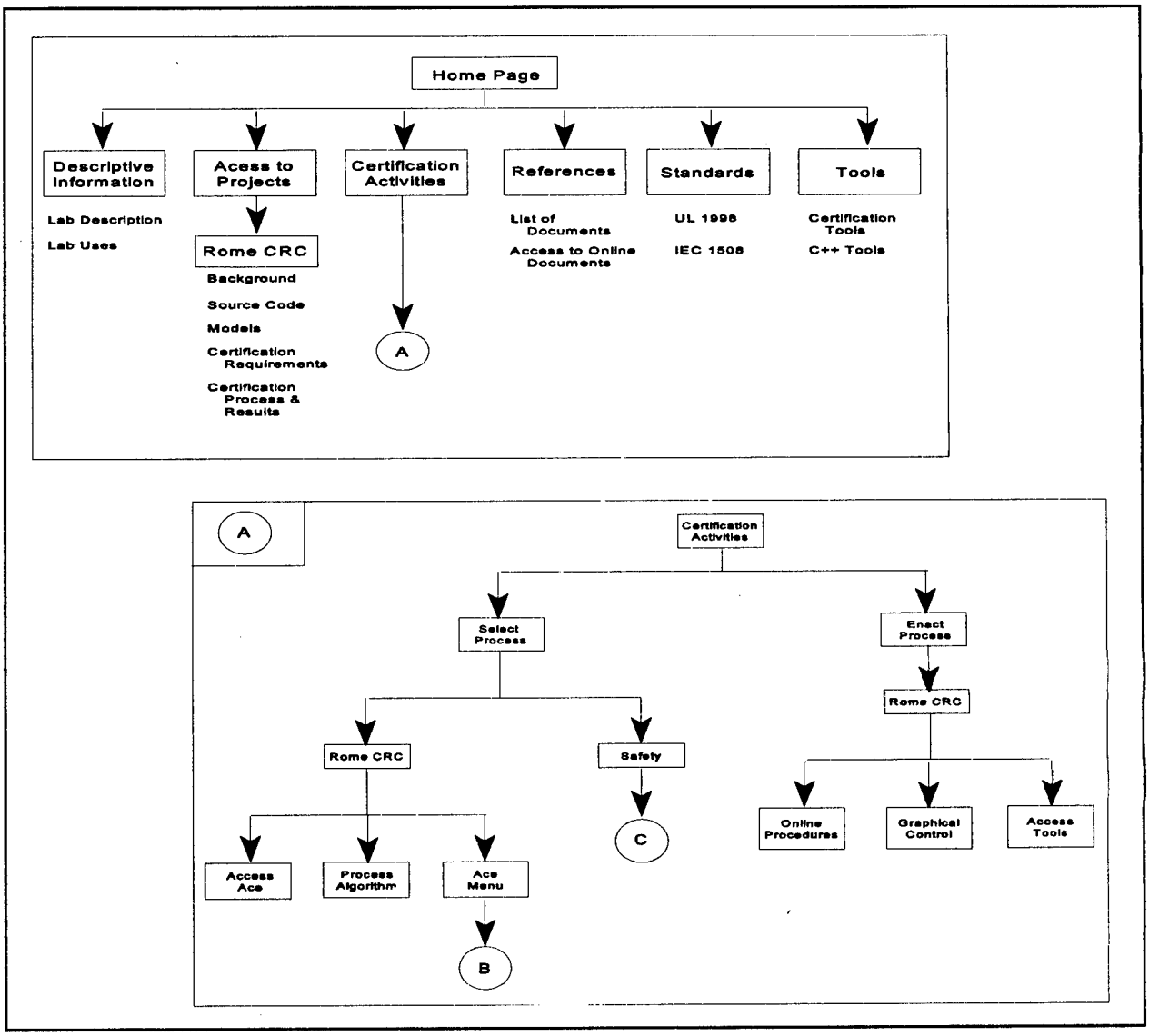

Figure 5.2: Web Interface

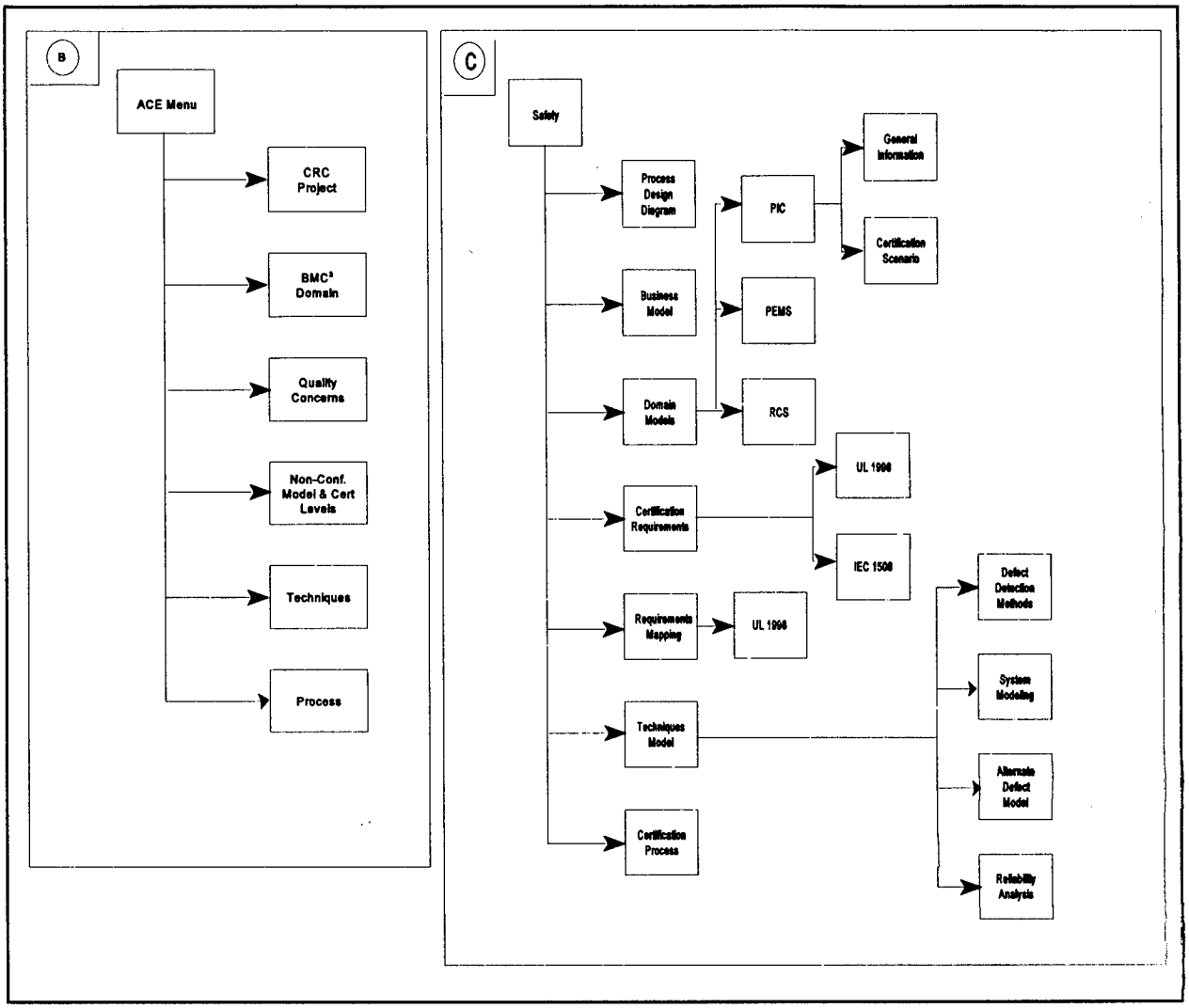

Figure 5.2 (Continued): Web Interface

The interface software allows the user to navigate through a suggested course of action, invoke testing tools on remote platforms, collect results to provide analysis on method effectiveness, etc., and generate reports. The application is well suited for navigation to be performed through hyperlinked text-based documents. Programs may be invoked from hyperlinked documents through tools such as Java, VB Script, etc. The application can also provide access to locally stored information databases on standards and methodologies as well as providing internet access to outside information sources. In the current prototype, only hyperlinked documents are implemented.

5.2 Outstanding Issues

Currently, the experimental lab provides access to all of the descriptive project files from the application of the framework and provides two options for creating certification processes in the form of two subtrees of the web. The first provides access to elements of the Rome framework and default process, including its defect detection model, certification level schema, and an online procedures guide. The second subtree provides an alternative means of assembling the elements of a certification process. It presents a graphical overview of the elements needed to create a certification process and links to particular instances of the elements, including domain models for safety applications, certification requirements extracted from safety standards, a mapping of requirements from a safety standard to the Rome framework scope levels and associated techniques, and sample defect detection technique models. Table 5.1 describes how the requirements of UL's software safety standard, UL 1998, map to the Rome framework. Part 1 of the table categorizes the types of defects, or faults, that UL 1998 addresses and shows the associated class(es) of defects and the range of applicable techniques from the framework. Part 2 of the table lists the types of analyses and procedures that are acceptable for meeting UL 1998 requirements and indicates the corresponding class(es) of techniques form the framework. As can be seen from this table, the framework does not address all of the elements required by the UL 1998 standard. In particular, hardware failures, analyses required to identify risks and critical sections of code, tool validation requirements, and certain life-cycle processes are not covered by the framework defect classes and techniques as currently specified. Some additional defect/technique models have been incorporated into the lab, but additional work is needed to identify and develop the models necessary to provide complete coverage of the UL 1998 standard.

Currently, the technique selection and ordering process specified in the framework contains a cost benefit optimization step that combines techniques based on cost benefit calculations. Whereas cost is an issue to all developers, it is not the primary concern in safety domains when decisions are made as to what has to be demonstrated and what are valid techniques to be used. The overriding factor in approving a technique is how effective it is. Also, techniques are usually specified by the life-cycle phase for which they are relevant and in which they can be used. The framework, with a focus on techniques being applied by the certifier, assumes that they are all applied at the same time; i.e., after the component has been completed and delivered. It is unclear whether defects that result from design mistakes, for example, can be found by evaluating the code and not the design. It is also not clear how the overlap in defects detected by different techniques is affected by applying them all at once rather than in sequence. This overlap, of course, is crucial to determining the effectiveness of combinations of techniques. Because of these issues with the selection decision process, we are using the lab to explore different decision methods and criteria, especially those that are based on the use of standards.

Table 5.1: Mapping of Methods in the Rome Laboratory Framework to UL 1998 Requirements

UL 1998		Rome Framework	
Part 1: Mapping of Methods			
Req No.	**Faults**	**Defect(s)**	**Techniques**
1.5.a	requirements conversion	latent	inspection, analysis, testing, formal proof
		robustness	inspection, analysis, simulation, formal proof
1.5.b	design		
6.1	no single-point failure	operational	analysis, testing, simulation, formal proof
6.2	return to RA state	operational	analysis, testing, simulation, formal proof
6.3	detect/handle software failures	operational	analysis, testing, simulation, formal proof
6.4	identify/respond to risky states	operational	analysis, testing, simulation, formal proof
6.5	risk-based scheduling	operational	analysis, testing, simulation, formal proof
6.6	prevent/detect/resolve Nterm/NDet/Err sts	latent/robustness	inspection, analysis, siml/tstng, formal proof
7.1	partitioning	latent/robustness	inspection, analysis, siml/tstng, formal proof
7.2	no memory usage/addressing conflicts	latent/robustness	inspection, analysis, siml/tstng, formal proof
7.3	control of sw by supervisory sw	operational	analysis, testing, simulation, formal proof
7.4	shutdown/fail op for failures of crit sw	operational	analysis, testing, simulation, formal proof
7.5	min 2 instr seqs to initiate risky functions	operational	analysis, testing, simulation, formal proof
7.6	initialization to known RA state	latent/robustness	inspection, analysis, siml/tstng, formal proof
7.7	controlled access to crit/sup instrs/data	latent/robustness	inspection, analysis, siml/tstng, formal proof
7.8	non-use of crit/sup instrs by non-c/s fns	latent/robustness	inspection, analysis, siml/tstng, formal proof
7.9	crit/sup sw to reside in non-volatile mem	latent/robustness	inspection, analysis, siml/tstng, formal proof
7.10	protect integrity of data used by crit/sup sw	operational	analysis, testing, simulation, formal proof
7.11	fixed/lmtd chng data stored in non-vol mem	latent/robustness	inspection, analysis, siml/tstng, formal proof
9	sw outputs to init hw to RA state	interoperability	inspection, analysis, simulation, formal proof
9.1	sw to transition to RA state if power failure	interoperability	inspection, analysis, simulation, formal proof
9.2	allocated init functions to be carried out	interoperability	inspection, analysis, simulation, formal proof
9.3	prod to transition to RA state if sw terminates	interoperability	inspection, analysis, simulation, formal proof
9.1	perm stop of cpu only if product in RA state	interoperability	inspection, analysis, simulation, formal proof
10.2a	no changes to parms that could affect intnd op	interoperability	inspection, analysis, simulation, formal proof
10.2b	config changes to parms to not affect intdn op	interoperability	inspection, analysis, simulation, formal proof
10.3	min 2 user responses to init risky operations	interoperability	inspection, analysis, simulation, formal proof
10.4	improper user input to not affect crit ops	interoperability	inspection, analysis, simulation, formal proof
10.5	provide user cancel & return to RA state	interoperability	inspection, analysis, simulation, formal proof
10.6	single input to cancel op	interoperability	inspection, analysis, simulation, formal proof
10.7	cancellation of proc to leave sw in RA state	interoperability	inspection, analysis, simulation, formal proof
1.5.c	coding	latent	inspection, analysis, testing, formal proof
		robustness	inspection, analysis, simulation, formal proof
1.5.d	timing	latent	inspection, analysis, testing, formal proof
		robustness	inspection, analysis, simulation, formal proof
		interoperability	inspection, analysis, simulation, formal proof
1.5.e	memory	interoperability	inspection, analysis, simulation, formal proof

1.5.f	hardware failures	operational	analysis, testing, simulation, formal proof
8.1 8.2a 8.2b 8.2c 8.2d 8.2e 8.2f 8.2g 8.2h 8.2i	measures for hw failure modes cpu regs, instr dec&exec, pgm ctr, addr/data paths interrupt handling and execution clock non-vol and vol memory & memory addressing internal data path & data addressing external communication - data, addressing, & timing I/O devices, such as analog I/O, D/A & A/D converters, analog multiplexors monitoring devices and comparators ASICs, GALs, PLAs, PGAs hardware		
1.5.g	state-dependent	robustness	inspection, analysis, simulation, formal proof
1.5.h	no function performed	operational	analysis, testing, simulation, formal proof
Part 2: Analyses/Procedures			
3.1	risk identification		
3.2	critical section identification.		
3.3	risky state identification		
4.1	quality management system		
5.1	tool validation		
11	software analysis and testing		
11.1	software code analysis		inspection, analysis, simulation, formal proof
11.1.1 11.1.2a 11.1.2b 11.1.2c 11.1.2d	performs only intended fns; no risk introduced correctness and completeness wrt spec decision criteria & function involving risk combs of sw, hw, and other events resulting in risk shutdown & fail-op procedures		
11.2	development and operational test		simulation, testing
11.2.1 11.2.2 11.2.3 11.2.4 11.2.5	test plan:parameters, procedures,criteria conduct tests; document test results test cases based on risk/code analy, safety ftrs test cases incl vals for parms where decs made effects on hw of sw outputs to be evaluated		

11.3	failure mode and stress testing		simulation, testing
11.3.1 11.3.2	testing under abnormal or off-nominal usage: operator errors component failures errs in data from external sensors or sw procs entry or exec failures of critical sections negative condition branch correct response & no risk to single-pt failures test cases incl vals for parms where decs made		
12	documentation		
12.1 12.2 12.3 12.4 12.5 12.6 12.7 12.8 12.9 12.10 12.11 6.7	design external interfaces operation & safety features wrt intended fn user documentation sw reference manual sw plan risk analysis approach & results configuration management plan system architecture plan system & software requirements specification system & software design specification information for third-party/OTS sw		
13	software changes		
13.1 13.2	no risk created, impacted, or increased lklhd procs to maintain control changes		
14	identification		
14.1 14.2	unique identifier in sw intended system configurations identified		

6 Findings

The Certification Framework is built upon clear, decisive and sound research. The Code Defect Model and Tool Selection can be substantiated by findings in previous research. However, while some of this research appears in the Field Trial Procedures, these procedures do not appear to be in accordance with the Code Defect Model. It is recommended that either the procedures be brought back in line with the framework or that justification be documented for their divergence. The Code Defect Model could also be expanded to address the impact of language differences and other sources of variation in defect type distribution such as differences in developers, application function, development testers, and variations in methods of computing lines of code.

The current framework has lost some of the applicability and flexibility of the initial design, especially in incorporating domain and application context into certification decisions. This particularly affects its use in certain areas, such as embedded, real-time applications and safety-critical applications. It is recommended that the program revisit the initial multi-level certification framework which enables a process by which a set of requirements is identified for the domain of applicability for which an asset is being developed and against which the asset is certified. It is also recommended that the mechanism proposed in the initial process for selecting an appropriate certification level be amended to incorporate the use of applicable standards. Standards would address process qualification, domain considerations, and criticality by product type and/or software functionality. The determination of applicable standards would make the selection mechanism objective and readily apparent.

It appears that there are several factors influencing how effective a technique is for finding defects. These factors include dependencies between conditions in different code segments, how much the control structure of the code changes with changes either in individual data items or in relationships between different data items, and whether a particular type of defect is more likely to be activated by particular control structures than others. Thus, it appears that the framework's effort to associate techniques with defect types is a necessary approach. However, it is also likely that more detailed analysis of the relationship between techniques and defects will be necessary before techniques can be recommended with confidence.

The framework, with a focus on techniques being applied by the certifier, assumes that they are all applied at the same time; i.e., after the component has been completed and delivered. It is unclear whether defects that result from design mistakes, for example, can be found by evaluating the code and not the design. It is also not clear how the overlap in defects detected by different techniques is affected by applying them all at once rather than in sequence. This overlap, of course, is crucial to determining the effectiveness of combinations of techniques.

Care was taken to construct a framework that is usable and practical. With some exceptions, the framework and the procedures are designed so that they can be used by a staff with minimal experience. They are practical for DOD repository use because they are very similar to the certification efforts currently in place. Since the certifier stops when the cost of certification becomes too high, the process is cost effective. The history of the repositories shows that only the least labor, time and cost intensive levels of certification will be used. It is not clear that the process as specified introduces any new or added value to the asset that current certification processes at the repositories don't. This is because the only way to provide that value and thereby benefit the user would be to take the asset to the highest level of certification - showing that it is functionally sound. This is not currently being done and, even though the need for it is demonstrated by the framework, the specified process does not make it any less costly or time consuming to accomplish.

Some specific recommendations for improving the default process involve regression testing, code inspection, and defect classification. The default process does not address regression. Correcting defects at every step is encouraged, but going back to the beginning of the certification process is not: only the current step is repeated. A fix in the code inspection step may make some code unreachable, for example, but since unreachable code was checked in the previous step, the new unreachable code would not be detected. In other words, the defect "fix" may have created one or more new defects that may not be detectable in the remaining or current steps. Because of the comprehensive nature of the checklist used in the code inspection step of the default process, the best results would be achieved if two or more inspectors reviewed the code, which is the case for most code reviews. The framework does not sufficiently differentiate between major and minor defects, leaving too much discretion to the certifier.

It is a significant challenge to a certifier, whether actually using a tool or evaluating its use by someone else, to know under what circumstances the tool is applicable and produces trustworthy results. This requires a detailed understanding of the underlying technique and an ability to discover the nuances of its implementation in the tool. To improve confidence in testing results, the tools used must be validated for the domain and context in which they are used. Based upon experience, the tools are not at a mature state where the test results they produce have a significant level of confidence.

The framework assumes that a user will consider code to be reusable even if it requires rework to be reused in a different application or domain from that for which it was originally developed. However, reusable assets in the commercial sector are not reworked. Attributes, such as color, size and font, are definable by the current user (application) but the inner workings are not presented for modification. Thus, the Certification Framework's selection of techniques based on rework avoidance is not applicable to a commercial asset. This discrepancy between the two

views of reuse can be a major obstacle to applying the framework to certify commercial assets. If the framework defines minimum criteria for reusable assets, any asset that cannot meet those criteria would be discarded or certified under a process for single use software.

While covering a broad spectrum, software certification has to be objective to be of value to the user of the certified software. Certification needs to be repeatable and consistent: any two certifying agencies should achieve the same results when applying the same certification standard to the same software component. Standardization of the results of certification based on the framework should be carefully considered. Currently, every certifying body would be able to use their own defect data and cost/benefit model to determine which techniques to use and how far to go in the process. The same asset, certified by two different agencies with different budgets and different goals for certification, would not be subjected to the same certification process in both places. A repository which places a higher emphasis on functional correctness and has no cost barriers may totally reject an asset that another repository may choose to certify only through the second level of the default process.

A certification framework can define how to selectively apply requirements to each software asset. Instead of the current certification level approach, it is recommended that a "meets standards" approach be utilized. A synergistic relationship between the framework and software certification standards is necessary to accomplish this goal. In specifying its requirements, the framework needs to reference existing standards that have been developed for different concerns, applications, and domains. Thus, the requirements for a particular asset would be tied to standards that apply to it. If there are key areas of the framework for which standards have not yet been defined, then the framework can identify where further research is needed and can establish requirements that would essentially be default standards. These default standards would be the starting point for new standards and the framework would be updated and revised as consensus emerged on the new standards.

A key finding from this activity is that the application of the procedures by a third party certifier on a component designed and produced for the commercial market may not be an appropriate application for the framework. It may be more practical for certifiers to use the framework to derive requirements for testing and evaluation of components that the developers would implement. The certifiers would then verify that the developers had implemented the specified activities. Thus, it is important that guidelines be developed to specify under what conditions a certifier would either perform the V&V activities defined in the framework or confirm that they were performed by the developer.

While the benefits of certification can perhaps be demonstrated to outweigh its costs under the current reuse library model, a greater cost/benefit could be more immediately evident under other

models, such as the Third Party Buyer-Supplier Model. In this model, standards are developed by consensus, quality is built in during development and costs are reduced, independence is maintained, cost of third party certification is reduced, and the third parties are accredited.

The opportunity exists for the Rome Laboratory Certification Framework to establish selection criteria and procedures for independent third party certifiers to use. The successful realization of this opportunity rests on certain assumptions, such as

- That a software parts supplier market can be created and will change how the software development industry operates,
- That independent certification will provide the confidence necessary for an application developer to buy a component rather than build or tailor one,
- That the Certification Framework can clearly establish the role it would play in the developing scenario for component certification, and
- That the present framework model can be enhanced to cover a broader view of the certification process.

Thus, a technical solution may not be as significant as the need for a business/market solution. Therefore, consideration should be given to focusing the framework for a buyer-supplier business model and to pursuing other modes of distribution than the traditional reuse repositories. An experimental lab such as that being established at UL would be a recommended approach to resolving the remaining technical issues, especially those related to technique effectiveness. The results of efforts in the lab will over time broaden the framework's certification model and evolve components of the framework so that they can be incorporated into certification standards.

References

[Clo94] John L. Cloninger. *What Would You Do If You Built A Library and Nobody Came?* 13 April 1994.

[RTI93] Research Triangle Institute. *Certification of Reusable Software Components*. U.S. Air Force Rome Laboratory Contract No. F30602-92-C-0158, March 1993.

[Sch91] Scheper, C., R. L. Baker, and H. L. Waters. *Integration of Tools for the Design and Assessment of High-Performance, Highly Reliable Computing Systems (DAHPHRS)*. Technical Report RL-TR-91-397, Rome Laboratory, December 1991.

[UL95] Sevio, B. *Investigation of the Market Opportunity for Independent Certification of Object-Oriented Components or Solutions*. Internal UL Report, March 1995.

☆U.S. GOVERNMENT PRINTING OFFICE: 1998-610-130-61122

MISSION OF ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Material Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.